Lustre® 1.4.6 Operations Manual

Version 1.4.6.1-man-v33

Cluster File Systems, Inc™

Lustre 1.4.6 Operations Manual

Version 1.4.6.1-man-v33 (08/01/2006)

This publication is intended to help Cluster File Systems, Inc. (CFS) Customers and Partners who are involved in installing, configuring, and administering Lustre.

Comments may be addressed to:

Cluster File Systems, Inc.

Suite E104 - 288

4800 Baseline Road

Boulder CO 80303

About the Manual

This Operations Manual is intended to support the users, architects and administrators of Lustre File Systems. It describes various tasks involved in installing, configuring, and administering Lustre.

For the ease of reference, the manual is sub-divided in various parts with respect to the audience they are meant for, as mentioned below –

1. Architecture – For Architects of Lustre
2. Lustre Administration – For System Administrators
3. Lustre Profiling, Monitoring and Troubleshooting - For System Administrators
4. Lustre for Users – For Users of Lustre
5. Reference – For all the three audiences, namely Architects, Users and System Administrators

TABLE OF CONTENTS

Conventions for Command Syntax

All the commands in this manual appear in green color of the font Courier New (point size 9) with the sign '| $' in the beginning. The other conventions are described below:

- Vertical Bar: '|' : To indicate alternative, mutually exclusive elements
- Square Brackets: '[ ]' : To indicate optional elements
- Braces: '{ }' : To indicate that a choice is required by the user
- Braces within brackets: '[{ }]' : To indicate that a choice is required within an optional element
- Backslash: ' \' : To indicate that the command line is continued on the next line
- **Boldface**: To indicate that the word is to be entered literally as shown
- *Italics*: To indicate a variable or argument to be replaced by the user with an actual value

Part I. Architecture

Chapter I – 1. A Cluster with Lustre

1.1 Lustre Server Nodes

A Lustre cluster consists of three major types of systems –

i) The Meta Data Server (MDS),

ii) The Object Storage Servers (OSS), and

iii) The Clients.

Each of these systems is internally very modular in the layout. For most of the modules like the locking module, the request processing layers and message passing layers are shared between all the systems. Thus, they form an integral part of the framework. The rest of the modules are unique. For example, the Lustre Lite client module on the client systems. Figure 1.1.1: 'A Lustre Cluster' gives an introductory impression of the expected interactions.

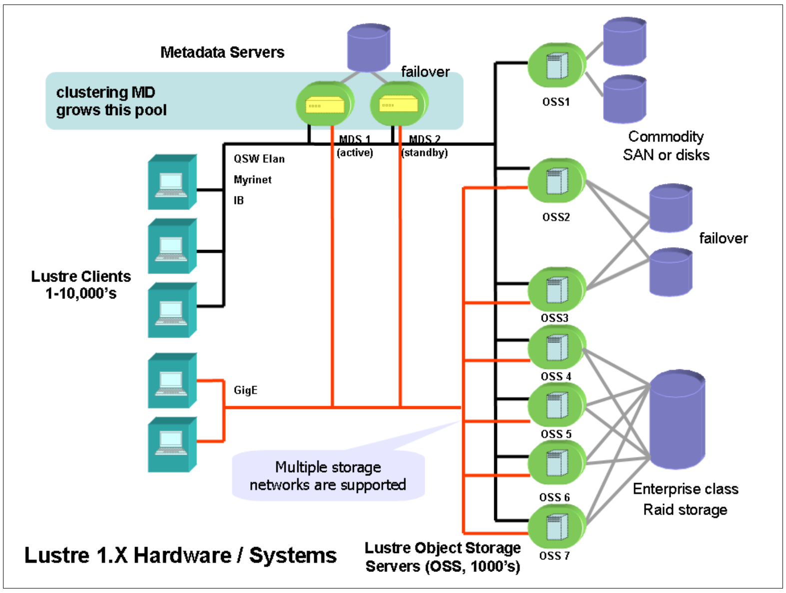

Figure 1.1.1: A Lustre Cluster

Lustre clients run the Lustre file system. They interact with the Object Storage Servers (OSSs) for file data I/O, and with the Meta Data Server (MDS) for name space operations.

When the Client, OSS, and MDS systems are separate, Lustre appears similar to a cluster file system with a file manager. But you can also have all these subsystems running on the same system, leading to a symmetrical layout. For the main protocols, see Figure 1.1.2: 'Interactions between the systems'.

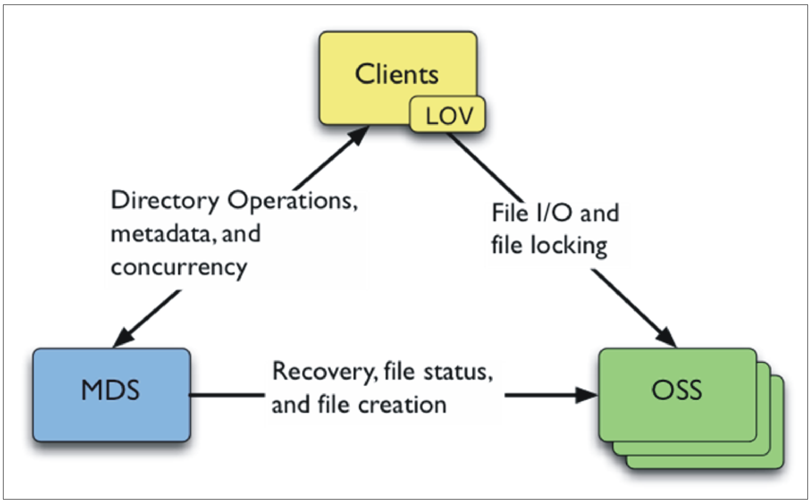

Figure 1.1.2: Interactions between the systems

1.1.1 MDS

The **meta data server** is perhaps the most complex subsystem. It provides back-end storage for the meta data service, and updates this with every transaction over a network interface. This storage presently uses a journal file system. Other options such as shared object storage are also considered. Figure 1.1.3: 'MDS Software Module' illustrates this function.

The MDS contains the locking modules, and heavily exercises the existing features of a journal file system, such as Ext3 or XFS. In Lustre, the complexity is limited due to the presence of a single meta data server. Yet the system avoids single points of failure by offering failover meta data services, based on existing solutions such as Linux-HA. In a Lustre with clustered meta data, meta data processing will be load balanced. This leads to a significant complexity in accessing the persistent meta data concurrently.

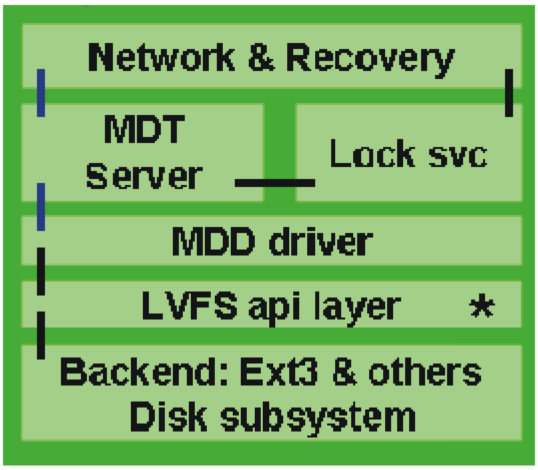

Figure 1.1.3: MDS Software Module

1.1.2 OSS

At the root of Lustre is the concept of **object storage**. Objects can be thought of as i-nodes. They are used to store file data. An Object Storage Server (OSS) is a server node running the Lustre software stack. It has one or more network interfaces, and usually one or more disks. Every OSS exports one or more Object Storage Targets (OST).

An Object Storage Target is a software interface to a single exported back-end volume. It is conceptually similar to an NFS export, except that an OST contains file system objects instead of the whole name space.

OSTs provide the file I/O service in a Lustre cluster by furnishing access to these objects. The name space is managed by meta data service, which manages the Lustre i-nodes. Such i-nodes can be directories, symbolic links, or special devices. In such cases, the associated data and meta data is stored on the meta data server. When a Lustre i-node represents a file, the meta data merely holds references to the file data objects stored on the OSTs.

The OSTs perform the block allocation for data objects, leading to distributed and scalable allocation of meta data. The OSTs also enforce security on the access to objects from clients. The client-OST protocol bears some similarity to systems like DAFS. It combines request processing with remote DMA. The software modules in the OSTs are indicated in Figure 1.1.4: 'OST Software Module'. Object storage targets provide a networked interface to other object storage. This second layer of object storage, so-called direct object storage drivers, consists of drivers that manage objects. These objects are the files on persistent storage devices. There are many choices for direct drivers, which are often interchangeable. Objects can be stored as raw ext2 i-nodes, or as files in many journal file systems by the filtering driver, which is now the standard driver for Lustre Lite. More exotic compositions of subsystems are possible. For example, in some situations an OBD filter direct driver can run on an NFS file system (a single NFS client is all that is supported).

In Figure 1.1.4: 'OST Software Module', networking is expanded into its subcomponents. Lustre request processing is built on a thin API, called the LNET API. LNET inter-operates with a variety of network transports through Network Abstraction Layers (NAL).

This API provides for the delivery and event generation in connection with network messages. It also provides advanced capabilities such as using Remote DMA (RDMA) if the underlying network transport layer supports this.

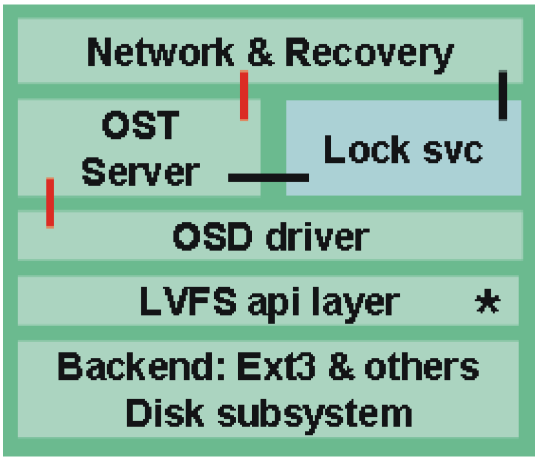

Figure 1.1.4: OST Software Module

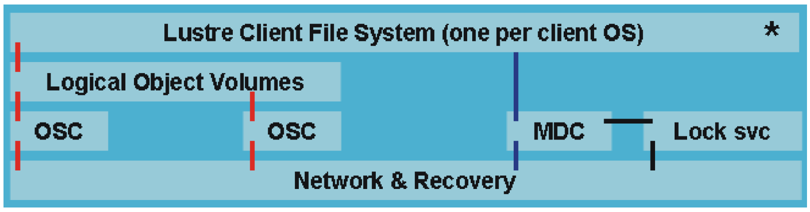

Figure 1.1.5: Client Software Module

Chapter I – 2. Understanding Lustre Networking

2.1 Introduction

Lustre now contains a new networking model known as LNET. LNET is designed for more complex topologies, better routing capabilities and simplified configuration. However, until a new configuration scheme is implemented for Lustre in 1.6.0, configuring LNET will require a hybrid of the old lconf/XML configuration and the new module-based configuration. Your patience through this transition is appreciated.

2.2 Old Schema

Up until Lustre 1.4.5, networking configuration under Portals was defined in the configuration XML file. Every node had a single NetworkID (NID) that Portals used to identify the communication endpoints, along with a specified network type (nettype). Different networks of the same type were specified with “cluster_id” attributes. A single NID could be bound to multiple network interfaces through “hostaddr” attributes. Routing was limited to Elan/TCP and defined in the XML in an over-complex way.

2.3 New Schema

The NID concept has been expanded to include the network upon which communication is taking place. There is now an NID for every network a node uses, and each NID includes the network type. Different networks of the same type are distinguished by a network number that is also a part of the NID. By default, LNET will use all the available interfaces for a specified network type, but all networking hardware specifics, including routing, are now defined as module options.

Due to the NID changes, old Portals and new LNET networking are NOT wire compatible. There is a provision for upgrading a live system, but this must be done in a particular order. See Chapter II – 8: 'Upgrading from 1.4.5'.

Network - A network is a group of nodes that communicate directly with each other. It is how LNET represents a single cluster. Multiple networks can be used to connect clusters together. Each network has a unique type and number (e.g. tcp3, elan1).

Network types include **tcp** (Ethernet), **openib** (Mellanox-Gold Infiniband), **iib** (Infinicon Infiniband), **vib** (Voltaire Infiniband), **ra** (RapidArray), **elan** (Quadrics Elan), **gm** (Myrinet), LNET.

NID - An NID is a Lustre networking address. Every node has one NID for each **network** it is on.

LND – Lustre Networking Device layer, a modular subcomponent of LNET that implements one of the network types.

Part II. Lustre Administration

CHAPTER II – 1. PREREQUISITES

1.1 Lustre Version Selection

1.1.1 How to get Lustre

The current, stable version of Lustre is available for download from the website of Cluster File Systems:

http://www.clusterfs.com/download.html

The software available for download on this website is released under the GNU General Public License. It is strongly recommended to read the complete license and release notes for this software before downloading it, if you have not done so already. The license and the release notes can also be found at the aforementioned website.

1.1.2 Supported Configurations

Cluster File Systems, Inc. supports Lustre on the configurations listed in Table1.1.1: 'Supported Configurations'.

ASPECT	SUPPORT TYPE
Operating Systems:	Red Hat Enterprise Linux 3+, SuSE Linux Enterprise Server 9, Linux 2.4 and 2.6
Platforms	IA-32, IA-64, x86-64, PowerPC architectures, and mixed-endian clusters
Interconnect	TCP/IP; Quadrics Elan 3 and 4; Myranet, Mellanox, Infiniband (Voltaire, OpenIB and Silverstrom)

Table 1.1.1: Supported Configurations

1.2 Using a Pre-packaged Lustre Release

Due to the complexity involved in building and installing Lustre, Cluster File Systems has made available several different pre-packaged releases that cover some of the most common configurations.

The pre-packaged release consists of five different RPM packages given below. Install them in the following order:

- **kernel-smp-<release-ver>.rpm** – This is the Lustre patched Linux kernel RPM. Use it with matching Lustre Utilities and Lustre Modules package.
- **kernel-source-<release-ver>.rpm** – This is the Lustre patched Linux kernel source RPM. This comes with the kernel package, but is not required to build or use Lustre.
- **lustre-modules-<release-ver>.rpm** – The Lustre kernel modules for the above kernel.
- **lustre-<release-ver>.rpm** – These are the Lustre Utilities or userspace utilities for configuring and running Lustre. Use them only with the matching kernel RPM as mentioned above.
- **lustre-source-<release-ver>.rpm** – This contains the Lustre source code (including the kernel patches). It is not required to build or use Lustre.

The source package is required only if you need to build your own modules (for networking, etc.) against the kernel source.

> NOTE: Lustre contains kernel modifications, which interact with your storage devices and may introduce security issues and data loss if not installed, configured, or administered properly. Please exercise caution and back up all data before using this software.

1.2.1 Choosing a Pre-packaged Kernel

Determining the best suitable pre-packaged kernel, depends largely on the combination of hardware and software being run. Cluster File Systems provides the pre-packaged releases listed in Table 1.2.1: 'Pre-packaged Release Details'.

RELEASE	DETAILS
1.4.6	Based on the 2.4.X Linux kernel, 2.6.X Linux Kernel and SusE Linux Enterprise for ia32, ia64 and x86_64 Platform, Supports both TCP and elan3

Table 1.2.1: Pre-packaged Release Details

1.2.2 Lustre Tools

The lustre * package is required for proper Lustre setup and monitoring. The package contains many tools, the most important ones being:

- **lconf**: High-level configuration tool that acts on XML files;
- **lctl**: A low-level configuration utility that can also be used for troubleshooting and debugging;
- **lfs**: A tool for reading/setting striping information for the cluster, as well as performing other actions specific to Lustre File Systems;
- **mount.lustre**: A mounting script required by Lustre clients.

1.2.3 Other Required Software

Besides the tools provided along with Lustre, Lustre also requires some separate software tools to be installed.

1.2.3.1 Core Requirements

Table 1.2.2: 'Software URLs' contains Hyperlinks to the software tools required by Lustre. Depending on your operating system, pre-packaged versions of these tools may be available, either from the sources listed below, or from your operating system vendor.

SOFTWARE	VERSION	LUSTRE FUNCTION
perl	>=5.6	Scripting language: used by monitoring and test scripts perl http://www.perl.com/download.csp
python	>=2	Scripting language: required by core Lustre tools python http://www.python.org/download/
PyXML	>=0.8	XML processor for python: requires PyXML http://sourceforge.net/project/showfiles.php?group_id=6473

Table 1.2.2: Software URLs

1.2.3.2 High Availability Software

If you plan to enable failover server functionality with Lustre (either on OSS or on MDS), a high availability software will be a necessary addition to your cluster software. One of the better known high availability software packages is Heartbeat.

Linux-HA (Heartbeat) supports redundant system with access to the Shared (Common) Storage with a dedicated connectivity; and can determine the general state of the system. (For details, see Part II - Chapter 5: 'Failover'.)

1.2.3.3 Debugging Tools

Things inevitably go wrong – disks fail, packets get dropped, software has bugs – and when they do, it is always useful to have debugging tools on hand to help figure out, how and why.

The most useful tool in this regard is GDB, coupled with crash. Together, these tools can be used to investigate both, live systems and kernel core dumps. There are also useful kernel patches/ modules, such as netconsole and netdump, that allow core dumps to be made across the network.

More information about these tools can be found at the following locations:

GDB: http://www.gnu.org/software/gdb/gdb.html

crash: http://oss.missioncriticallinux.com/projects/crash/

netconsole: http://lwn.net/2001/0927/a/netconsole.php3

netdump: http://www.redhat.com/support/wpapers/redhat/netdump/

1.3 Environment Requirements

1.3.1 Consistent Clocks

Lustre always uses the client clock for timestamps. If the machine clocks across the cluster are not in sync, Lustre should not break. However, the unsynchronized clocks in a cluster will always be a source of headache as it will be very difficult to debug any multi-node issue, or otherwise correlate the logs. For this reason, CFS recommends that the machine clocks should be kept in sync as much as possible. The standard way to accomplish this is by using the Network Time Protocol, or NTP. All the machines in your cluster should synchronize their time from a local time server (or servers) at a suitable time interval.

More information about ntp can be found at:

http://www.ntp.org/

1.3.2 Universal UID/GID

In order to maintain uniform file access permissions on all the nodes of your cluster, the same user (UID) and group (GID) IDs should be used on all the clients. Pretty much like any cluster usage, Lustre uses the common UID/GID on all the cluster nodes.

1.3.3 Proper Kernel I/O Elevator

The Linux kernel offers a choice of I/O schedulers. The I/O scheduler is also known as 'the elevator'. There are currently four choices available:

- deadline – This is the 'old' scheduler, which works well for servers.
- anticipatory I/O scheduler (AS) – It is designed for 'batching' I/O requests. It does not work well for servers and high IO loads.
- cfq – It adds multiple scheduling classes. cfq also does not work well for servers and high I/O loads.
- noop – This is the 'old' elevator. It works well for servers.

To ensure good I/O throughput on Lustre servers, we recommend that the default I/O scheduler should be changed to one of the following options -

i) deadline

ii) noop

There are two ways to change the I/O scheduler - at boot time, or with new kernels at runtime. For all Linux kernels, appending 'elevator={noop|deadline}' to the kernel boot string sets the I/O elevator.

With LILO, you can use the 'append' keyword:

```
image=/boot/vmlinuz-2.6.14.2
label=14.2
append="elevator=deadline"
read-only
optional
```

With GRUB, append the string to the end of the kernel command:

```
title Fedora Core (2.6.9-5.0.3.EL_lustre.1.4.2custom)
root (hd0,0)
kernel /vmlinuz-2.6.9-5.0.3.EL_lustre.1.4.2custom ro
root=/dev/VolGroup00/LogVol00 rhgb noapic quiet
elevator=deadline
```

With newer Linux kernels (Red Hat Enterprise Linux v3 Update 3 does not have this feature. It is present in the main Linux tree as of 2.6.15), one can change the scheduler while running. If the file /sys/block/<DEVICE>/queue/scheduler exists (where DEVICE is the block device you wish to affect), it will contain a list of available schedulers and can be used to switch the schedulers.

(hda is the <disk>):

```
[root@cfs2]# cat /sys/block/hda/queue/scheduler
noop [anticipatory] deadline cfq
[root@cfs2 ~]# echo deadline >
/sys/block/hda/queue/scheduler
[root@cfs2 ~]# cat /sys/block/hda/queue/scheduler
noop anticipatory [deadline] cfq
```

The other schedulers (anticipatory and cfq) are better suited for desktop use.

Chapter II – 2. Lustre Installation

2.1 Installing Lustre

Follow the steps given below to install Lustre:

1. Install the OS as per your requirements along with the Prerequisites like GCC, Python and Perl (as mentioned in Chapter II – 1. Prerequisites).

2. Then install the RPMs as described in section 1.2 'Using a Pre-packaged Lustre Release' in Chapter II – 1. Prerequisites. The preferred installation order is:

- The Lustre patched version of the linux kernel (kernel-*)
- The Lustre modules for that kernel (lustre-modules-*)
- The Lustre user space programs (lustre-*)
- The other packages are optional.

3. Verify all cluster ntetworking is correct. This may include /etc/hosts, or DNS. Set the correct networking options for Lustre in /etc/modprobe.conf. (See 5.1.1 and 5.2.2 'Modprobe.conf' in Chapter II – 5. 'More Complicated Configurations'.)

> **TIP:**
> While installing Lustre with InfiniBand, you need to keep the ibhost, kernel and Lustre all on the same revision. Follow the steps below to achieve this:
> 1. Install the kernel source (Lustre patched)
> 2. Install the Lustre source and the ibhost source
> 3. Compile the ibhost against your kernel
> 4. Compile the Linux kernel
> 5. Compile Lustre against the ibhost source --with-vib=<path to ibhost>
> Now you can use the RPMs created by the above steps.

2.2 Quick Configuration of Lustre

As we have already seen, Lustre consists of 3 types of subsystems - a Metadata Server (MDS), Object Storage Targets (OSTs), and Clients. All these can co-exist on a single system or can run on different systems. A Lustre client system can also optionally have a Logical Object Volume manager (LOV) to transparently manage several OSTs. This component is required for achieving file striping.

It is possible to set up the Lustre system with many different configurations by using the administrative utilities provided with Lustre. Lustre includes some sample scripts in the */usr/lib/lustre/examples* directory on a system where Lustre is installed (or the *lustre/tests* subdirectory of a source code installation). These scripts enable quick setup of some simple, standard configurations.

Let us see how to install a simple Lustre setup using these scripts.

2.2.1 Single System Test Using the llmount.sh Script

The simplest Lustre installation is a configuration where all three subsystems execute on a single node. You can execute the script llmount.sh, located in the /usr/lib/lustre/examples directory, to set up, initialize, and start the Lustre file system on a single node. This script first executes a configuration script identified by a 'NAME' variable. This configuration script uses the LMC utility to generate an XML configuration file, which is in turn used by the LCONF utility to perform the actual system configuration. The llmount.sh script then loads all the modules required by the specified configuration.

Next, the script creates small loopback file systems in /tmp for the server nodes. You can change the size and location of these files by modifying the configuration script.

Finally, the script mounts the Lustre file system at the mount point specified in the initial configuration script. The default used is /mnt/lustre. Given below are the steps needed to configure and test Lustre for a single system.

(You can use the llmount.sh script for initial testing to hide many of the background steps needed to configure Lustre. It is not intended to be used as a configuration tool for production installations.)

A. Starting the system: Two initial configuration scripts are provided for a single system test. In order to start the system, update these scripts as per the changes made to the loopback file system locations or sizes, or as per the changes made to the Lustre file system mount point.

- Execute the local.sh script having the LMC commands to generate an XML (local.xml) configuration file for a single system.
- Execute the lov.sh script having the LMC commands to generate a configuration file with an MDS, LOV, two OST's and a Client.

B. Execute the llmount.sh script as shown below, specifying the setup based on either local.sh or lov.sh:

```
$ NAME={local|lov} sh llmount.sh
```

Below is a sample output of executing this command on a random system:

```
$ NAME=local sh llmount.sh

loading module: libcfs srcdir None devdir libcfs

loading module: lnet srcdir None devdir lnet

loading module: ksocklnd srcdir None devdir \
klnds/socklnd

loading module: lvfs srcdir None devdir lvfs

loading module: obdclass srcdir None devdir obdclass

loading module: ptlrpc srcdir None devdir ptlrpc

loading module: ost srcdir None devdir ost

loading module: ldiskfs srcdir None devdir ldiskfs

loading module: fsfilt_ldiskfs srcdir None devdir \
lvfs

loading module: obdfilter srcdir None devdir \
obdfilter

loading module: mdc srcdir None devdir mdc

loading module: osc srcdir None devdir osc

loading module: lov srcdir None devdir lov

loading module: mds srcdir None devdir mds

loading module: llite srcdir None devdir llite

NETWORK: NET_mds.clusterfs.com_tcp \
NET_mds.clusterfs.com_tcp_UUID tcp \
mds.clusterfs.com

OSD: OST_mds.clusterfs.com \
OST_mds.clusterfs.com_UUID obdfilter /tmp/ost1- \
mds.clusterfs.com 400000 ldiskfs no 0 0

OST mount options: errors=remount-ro

MDSDEV: mds1 mds1_UUID /tmp/mds1-mds.clusterfs.com \
ldiskfs no

recording clients for filesystem: FS_fsname_UUID

Recording log mds1 on mds1

LOV: lov_mds1 110aa_lov_mds1_3af7a3d69c mds1_UUID 1 \
1048576 0 0 [u'OST_mds.clusterfs.com_UUID'] mds1 \
```

```
OSC: \
OSC_mds.clusterfs.com_OST_mds.clusterfs.com_mds1 \
110aa_lov_mds1_3af7a3d69c OST_mds.clusterfs.com_UUID

End recording log mds1 on mds1

MDSDEV: mds1 mds1_UUID /tmp/mds1-mds.clusterfs.com \
ldiskfs 400000 no

MDS mount options: errors=remount-ro,user_xattr,acl,

LOV: lov1 4987a_lov1_765ed779f4 mds1_UUID 1 1048576 \
0  0 [u'OST_mds.clusterfs.com_UUID'] mds1

OSC: \
OSC_mds.clusterfs.com_OST_mds.clusterfs.com_MNT_mds \
.clusterfs.com 4987a_lov1_765ed779f4 \
OST_mds.clusterfs.com_UUID

MDC: \
MDC_mds.clusterfs.com_mds1_MNT_mds.clusterfs.com \
f9c37_MNT_mds.clusterfs._30aaf9b569 mds1_UUID

MTPT: MNT_mds.clusterfs.com \
MNT_mds.clusterfs.com_UUID /mnt/lustre mds1_UUID \
lov1_UUID
```

Now you can verify if the file system is mounted from the output of df:

```
$ df

Filesystem  1K-blocks  Used  Available  Use%   \
Mounted on

/dev/ubd/0   1011928   362012   598512   38%  /

/dev/ubd/1   6048320   3953304  1787776  69%  /r

none         193712    16592    167120   10% \
/mnt/lustre
```

> NOTE: The output of the df command following the output of the script shows that the Lustre file system is mounted on the mount-point /mnt/lustre. The actual output of the script included with your Lustre installations may have changed due to enhancements or additional messages, but should resemble the example.

You can also verify that the Lustre stack has been set up correctly by observing the output of find /proc/fs/lustre.

> NOTE: The actual output may depend on the inserted modules and on the instantiated OBD devices. Also note that the file system statistics presented from /proc/fs/lustre are expected to be the same as those obtained from df.

C. You can bring down a cluster and clean it up by using the llmountcleanup.sh script. Execute the command below to cleanup and unmount the file system:

```
$ NAME=<local/lov> sh llmountcleanup.sh
```

D. To remount the file system, you can use the llrmount.sh script. Using llmount.sh again reformats the devices. Hence, use llrmount.sh if you want to retain data in the file system.

```
$ NAME=<local/lov> sh llrmount.sh
```

2.3 Using Supplied Configuration Tools

It is possible to set up Lustre on either a single system or on multiple systems. The Lustre distribution comes with utilities that can be used to create configuration files easily, and to set up Lustre for various configurations. Lustre uses three administrative utilities – LMC, LCONF and LCTL – to configure nodes. The LMC utility is used to create XML configuration files describing the configuration. The LCONF utility uses the information in this configuration file to invoke the low-level configuration utility LCTL. LCTL in turn, actually configures the systems. For further details on these utilities please refer the man pages. Keep the complete configuration for the whole cluster in a single file. Use the same file on all the cluster nodes to configure the individual nodes.

The next few sections describe the process of setting up a variety of configurations.

> **TIP:** You can use "lconf -v" to show more verbose messages when running other LCONF commands.

> NOTE: You must use fstype = ext3 for Linux 2.4 kernels, and fstype = ldiskfs for 2.6 kernels. (In 2.4, Lustre patches the ext3 driver while in 2.6, it provides its own driver.)

2.3.1 Single Node Lustre

Let us consider a simple configuration script where the MDS, the OSTs, and the client are running on a single system. You can use the LMC utility to generate a configuration file for this as shown below. All the devices in the script are shown to be loopback devices, but you can specify any device here. The size option is required only for the loopback devices; for others the utility will extract the size from the device parameters. (See the usage of Real Disks below).

```
$ !/bin/sh

$ local.sh
```

To create a node:

```
rm -f local.xml

lmc -m local.xml --add node --node localhost

lmc -m local.xml --add net --node localhost -nid \
localhost --nettype tcp
```

To configure the MDS:

```
lmc -m local.xml --format --add mds --node localhost\
--mds mds-test --fstype ext3 --dev /tmp/mds-test  \
-size 50000
```

To configure the OSTs:

```
lmc -m local.xml --add lov --lov lov-test --mds mds-\
test  --stripe_sz 1048576 --stripe_cnt 0 --\
stripe_pattern 0

lmc -m local.xml --add ost --node localhost -lov \
lov-test --ost ost1-test --fstype ext3 -dev \
/tmp/ost1-test --size 100000

lmc -m local.xml --add ost --node localhost -lov \
lov-test --ost ost2-test --fstype ext3 -dev \
/tmp/ost2-test --size 100000
```

To configure the client:

```
lmc -m local.xml --add mtpt --node localhost -path \
/mnt/lustre --mds mds-test --lov lov-test
```

On running the script, these commands create a *local.xml* file describing the specified configuration. Now you can execute the actual configuration by using the following command:

To configure using LCONF:

```
$ sh local.sh

$ lconf --reformat local.xml
```

This command loads all the required Lustre and Portals modules, and also does the low level configuration of every device using LCTL. The reformat option here is essential at least the first time to initialize the file systems on the MDS and OSTs. If it is used on any subsequent attempts to bring up the Lustre system, it will re-initialize the file systems.

2.3.2 Multiple Node Lustre

Now let us see an example of setting up Lustre on multiple systems – with the MDS on one node, the OSTs on other nodes, and the client on one or more nodes. You can use the following configuration script to create such a setup. Replace *node-** in the example with the hostnames of real systems. The servers, clients, and the node running the configuration script all need to resolve those hostnames into IP addresses via DNS or /etc/hosts. One common problem with some Linux setups is that the hostname is mapped in /etc/hosts to 127.0.0.1, which causes the clients to fail in communicating with the servers.

1. Create nodes using the LMC command:

```
rm -f config.xml

lmc -m config.xml --add net --node node-mds --nid \
node-mds --nettype tcp

lmc -m config.xml --add net --node node-ost1 --nid \
node-ost1 --nettype tcp

lmc -m config.xml --add net --node node-ost2 --nid \
node-ost2 --nettype tcp
```

```
lmc -m config.xml --add net --node node-ost3 --nid \
node-ost3 --nettype tcp

lmc -m config.xml --add net --node client --nid '*' \
--nettype tcp
```

2. Configure the MDS with the LMC command:

```
lmc -m config.xml --add mds --node node-mds --mds \
mds-test --fstype ext3 --dev /tmp/mds-test --size
50000
```

3. Configure the OSTs with the LMC command:

```
lmc -m config.xml --add lov --lov lov-test --mds \
mds-test --stripe_sz 1048576 --stripe_cnt 0 --\
stripe_pattern 0

lmc -m config.xml --add ost --node node-ost1 --lov \
lov-test --ost ost1-test --fstype ext3 --dev /tmp/ \
ost1-test --size 100000

lmc -m config.xml --add ost --node node-ost2 --lov \
lov-test --ost ost2-test --fstype ext3 --dev /tmp/ \
ost2-test --size 100000

lmc -m config.xml --add ost --node node-ost3 --lov \
lov-test --ost ost3-test --fstype ext3 --dev /tmp/ \
ost3-test --size 100000
```

4. Configure the client (a 'generic' client used for all the client mounts):

```
lmc -m config.xml --add mtpt --node client -path \
/mnt/lustre --mds mds-test --lov lov-test
```

5 Generate the config.xml (once). Put the file at a location where all the nodes can access it. For example, an NFS Share.

2.3.3 Starting Lustre

Follow the steps below to start Lustre:

A .Start the OSTs first:

```
$ lconf --reformat --node node-ost1 config.xml

$ lconf --reformat --node node-ost2 config.xml

$ lconf --reformat --node node-ost3 config.xml
```

B. Start the MDS (which tries to connect to the OSTs):

```
$ lconf --reformat --node node-mds config.xml
```

C. Start the clients (which try to connect to the OSTs and the MDS):

```
$ lconf --node client config.xml

$ mount -t lustre node-mds:/mds-test/client
/mnt/lustre
```

2.4 Building From Source

2.4.1 Building Your Own Kernel

In case the hardware is not the standard hardware or CFS support have asked to apply a patch, then Lustre requires a few changes to the core Linux kernel. These changes are organized in a set of patches in the kernel_patches directory of the Lustre CVS repository. If you are building your kernel from the source, you will need to apply the appropriate patches.

Managing patches for the kernels is a very involved process, and most patches are intended to work with several kernels. To facilitate the support, CFS maintains the tested version on the FTP site as some versions may not work properly with the patches from CFS. The Quilt package developed by Andreas Gruenbacher simplifies the process considerably. The patch management with Quilt works as follows:

- A series file lists a collection of patches;
- The patches in a series form a stack;
- One can push and pop the patches with Quilt;
- One can edit and refresh (update) the patches if the stack is being managed with Quilt;
- One can revert inadvertent changes and fork or clone the patches and conveniently show the difference in work, before and after.

2.4.1.1 Patch Series Selection

Depending on the kernel being used, a different series of patches needs to be applied. Cluster File Systems maintains a collection of different patch series files for the various supported kernels in lustre/kernel_patches/series/.

For instance, the file lustre/kernel_patches/series/rh-2.4.20 lists all the patches that should be applied to a Red Hat 2.4.20 kernel to build a Lustre compatible kernel.

The current set of all the supported kernels and their corresponding patch series can always be found in the file –

lustre/kernel_patches/which_patch.

2.4.1.2 Using Quilt

A variety of Quilt packages (RPMs, SRPMs, and tarballs) are available are available from Linux. As Quilt changes from times to time, we advise you to download the appropriate package from the ftp site of Cluster File Systems:

ftp://ftp.clusterfs.com/pub/quilt/

The Quilt RPMs have some installation dependencies on other utilities, e.g. the core-utils RPM that is available only in Red Hat 9. You will also need a recent version of the diffstat package. If you cannot fulfill the Quilt RPM dependencies for the packages made available by Cluster File Systems, we suggest building Quilt from the tarball.

After you have acquired the Lustre source (CVS or tarball), and chosen a series file to match you kernel sources, you must also choose a kernel config file. The supported kernel ".config" files are in the folder lustre/kernel_patches/kernel_configs, and are named in such a way as to indicate which kernel and architecture they are meant for. E.g. vanilla-2.4.20.uml.config is a UML config file for the vanilla 2.4.20 kernel series.

Next, unpack the appropriate kernel source tree. For the purposes of illustration, this documentation assumes that the resulting source tree is in /tmp/kernels/linux-2.4.20 and we call this the destination tree.

You are now ready to use Quilt to manage the patching process for your kernel. The following set of commands will setup the necessary symlinks between the Lustre kernel patches and your kernel sources.

```
$ cd /tmp/kernels/linux-2.4.20
$ quilt setup
-l ../lustre/kernel_patches/series/rh-2.4.20 \
-d ../lustre/kernel_patches/patches
```

You can now have Quilt apply all the patches in the chosen series to your kernel sources by using the set of commands given below.

```
$ cd /tmp/kernels/linux-2.4.20
$ quilt push -av
```

If the right series files are chosen and the patches and the kernel sources are up-to-date, the patched destination Linux tree should now be able to act as a base Linux source tree for Lustre.

The patched Linux source need not be compiled to build Lustre from it. However, the same Lustre-patched kernel must be compiled and then booted on any node on which you intend to run a version of Lustre built using this patched kernel source.

2.4.2 Building Lustre

The Lustre source can be obtained by registering on the site -

http://www.clusterfs.com/download.html

You will receive a mail with the link for download.

The following set of packages are available for each supported

Linux distribution and architecture. The files employ the naming convention:

kernel-smp-<kernel versrion>_lustre.<lustre version>.<arch>.rpm

- Example of **binary packages** for 1.4.6:

- kernel-smp-2.6.9-22.0.2.EL_lustre.1.4.6.i686.rpm will contain patched kernel
- lustre-1.4.6-2.6.9_22.0.2.EL_lustre.1.4.6smp.i686.rpm will contain Lustre user space files and utilities
- lustre-modules-1.4.6-2.6.9_22.0.2.EL_lustre.1.4.6smp.i686.rpm will contain Lustre modules (kernel/fs/lustre and kernel/net/lustre)

You can install the binary package by issuing the standard rpm commands –

```
$ rpm -ivh kernel-smp-2.6.9- \
22.0.2.EL_lustre.1.4.6.i686.rpm

$ rpm -ivh lustre-1.4.6- \
2.6.9_22.0.2.EL_lustre.1.4.6smp.i686.rpm

$ rpm -ivh lustre-modules-1.4.6-2.6.9_22.0.2.EL_ \
lustre.1.4.6 smp.i686.rpm
```

- Example of **Source packages**:

- kernel-source-2.6.9-22.0.2.EL_lustre.1.4.6.i686.rpm will contain the source for the patched kernel
- lustre-source-1.4.6-2.6.9_22.0.2.EL_lustre.1.4.6smp.i686.rpm will contain the source for Lustre modules and user space utilities

The kernel-source and lustre-source packages are provided in case you need to build external kernel modules or use additional network types. They are not required to run Lustre.

Once you have your Lustre source tree, you can build Lustre by running the sequence of commands given below.

```
$ cd <path to kernel tree>

$ cp /boot/config-'uname -r' .config

$ make oldconfig || make menuconfig
```

For 2.6 kernels

```
  $ make include/asm

  $ make include/linux/version.h

  $ make SUBDIRS=scripts
```

For 2.4 kernels

```
  $ make dep
```

To configure Lustre, and to build Lustre RPMs go into the Lustre source directory, and run:

```
$ ./configure --with-linux=<path to kernel tree>

$ make rpms
```

This will create a set of .rpms in /usr/src/redhat/RPMS/<arch>

with a date-stamp appended. (The SUSE path is /usr/src/packages)

Example:

```
lustre-1.4.6-2.6.9-22.0.2.EL_lustre.1.4.6custom_ \
200507072009.i386.rpm

lustre-debuginfo-1.4.2-2.6.9-\
22.0.2.EL_lustre.1.4.6custom_200507072009.i386.rpm

lustre-modules-1.4.2-2.6.9-\
22.0.2.EL_lustre.1.4.6custom_200507072009.i386.rpm

lustre-source-1.4.2-2.6.9-\
22.0.2.EL_lustre.1.4.6custom_200507072009.i386.rpm
```

cd into the kernel source directory, and run

```
$ make rpm
```

This will create a kernel rpm suitable for the installation.

Example: kernel-2.6.95.0.3.EL_lustre.1.4.2custom-1.i386.rpm

2.4.2.1 Configuration Options

Lustre supports several different features and packages that extend the core functionality of Lustre. These features/packages can be enabled at the build time by issuing appropriate arguments to the configure command. A complete listing of the supported features and packages can always be obtained by issuing the command "./configure –help" in your Lustre source directory. The config files matching the kernel version are in the configs/ directory of the kernel source. Copy one to .config at the root of the kernel tree.

2.4.2.2 Liblustre

The Lustre library client, liblustre, relies on libsysio, which is a library that provides POSIX-like file and name space support for remote file systems from the application program address space. Libsysio can be obtained from:

http://sourceforge.net/projects/libsysio/

> NOTE: Liblustre is not for general use. It was created to work with specific hardware (Cray) and should never be used with other hardware.

Development of libsysio has continued ever since it was first targeted for use with Lustre. So first checkout the b_lustre branch from the libsysio CVS repository. This gives the version of libsysio compatible with Lustre. Once checked out, the steps listed below will build libsysio.

```
$ sh autogen.sh

$ ./configure --with-sockets

$ make
```

Once libsysio is built, you can build liblustre using the following commands.

```
$ ./configure --with-lib -with-sysio=/path/to/
libsysio/source
$ make
```

2.4.2.3 Compiler Choice

The compiler must be greater than GCC version 3.3.4. GCC v4.0 is not currently supported. GCC v3.3.4 has been used to successfully compile all of the pre-packaged releases made available by Cluster File Systems, and as such is the only compiler that is officially supported. Your mileage may vary with other compilers, or even other versions of GCC.

> NOTE: GCC v3.3.4 was used to build 2.6 series kernels.

Chapter II – 3. Configuring the Lustre Network

3.1 Designing Your Network

Before configuration can take place, a clear understanding of your Lustre network topologies is essential.

3.1.1 Identify all Lustre Networks

A network is a group of nodes that communicate directly with each other. As mentioned previously, Lustre supports a variety of network types and hardware, including TCP/IP, Elan, varieties of Infiniband and others. The normal rules for specifying networks apply, for example, two TCP networks on two different subnets would be considered two different Lustre networks. For example, tcp0 and tcp1.

3.1.2 Identify nodes which will route between networks

Any node with appropriate interfaces can route LNET between different networks – the node may be a server, a client, or a standalone router. LNET can route across different network types (For example, TCP to Elan) or across different topologies (For example, bridging two Infiniband or TCP/IP networks).

3.1.3 Identify any network interfaces that should be included/excluded from Lustre networking

LNET by default uses all interfaces for a given network type. If there are interfaces it should not use, (e.g. Administrative networks, IP over IB, etc), then the included interfaces should be explicitly listed.

3.1.4 Determine cluster-wide module configuration

The LNET configuration is managed via module options, typically specified in /etc/modprobe.conf or /etc/modprobe.conf.local (depending on distro). To help ease the maintenance of large clusters, it is possible to configure the networking setup for all nodes through a single unified set of options in the modprobe.conf file on each node. See the ip2nets option below for more information.

LibLustre users should set the accept=all parameter, see the appendix for details.

3.1.5 Determine appropriate zconf-mount parameters for clients

In their mount commands, clients use the NID of the MDS host to retrieve their configuration information. Since an MDS may have more than one NID, clients should use the NID appropriate for its local networks. If unsure, there is a 'lctl' command that can help. On the MDS,

```
lctl list_nids
```

will display the server's NIDs. On a client,

```
lctl which_nid <NID list>
```

will display the closest NID for that client. So from a client with SSH access to the MDS,

```
mds_nids=`ssh the_mds lctl list_nids`

lctl which_nid $mds_nids
```

will in general be the correct NID to use for the MDS in the mount command.

3.2 Configuring Your Network

LNET before mountconf (version 1.4.6 and 1.4.7)

3.2.1 LNET Configurations

LNET and portals use different network addressing schemes; i.e. their NIDs are different. Lmc/lconf allow NIDs to be specified in either format so that old configuration (lmc) scripts and old XML configuration files continue to work and the NIDs are converted to LNET format as required.

LNET NIDs take the form: *nid = <address>[@<network>]*, where <address> is the network address within the network and <network> is the identifier for the network itself (network type + instance number). For example, 192.73.220.107@tcp0 would be a typical NID on a TCP network. '3@elan0' would be a typical Elan NID.

The network number can be used to distinguish between instances of the same network type, e.g. tcp0 and tcp1. An unspecified network number is '0', and unspecified network type is 'tcp'.

> NOTE: If a machine has multiple network interfaces, Lustre networking must be specified by modprobe.conf options (networks or ip2nets) as the default configuration will almost certainly not work for a multi-homed host.

3.2.1.1 NID Changes

The LNET NID is generated from old (lmc) configuration scripts by using the network type (specified by '--nettype <type>') as the LNET network identifier. New configuration scripts should use the network type 'lnet' and specify the LNET NID directly.

Example lmc line specifying a server's NID:

```
$ LMC --add net --node srv1 --nettype lnet --nid
192.168.2.1@tcp1
```

(The lmc tool will be obsolete with mountconf in Lustre 1.6.0)

Example lmc line for clients on all networks:

```
$ LMC --add net --node client --nettype lnet --nid
'*'
```

A client's actual NIDs are determined from its local networks at client startup time.

3.2.1.2 XML Changes

These changes affect lmc and the XML it produces, as well as zeroconf mount commands. (The lmc tool will be obsolete with mountconf in Lustre 1.6.0)

Example zeroconf client mount command pointing to an MDS on an elan network:

```
mount -t lustre 3@elan:/mdsA/client /mnt/lustre
```

NOTE: We recommend using dotted-quad IP addressing rather than host names. We have found this aids in reading debug logs, and helps greatly when debugging configurations with multiple interfaces.

3.2.2 Module parameters

LNET network hardware and routing are now configured via module parameters of the LNET and LND-specific modules. Parameters should be specified in the /etc/modprobe.conf or /etc/modules.conf file, e.g.:

```
options lnet networks=tcp0,elan0
```

specifies that this node should use all available TCP and elan interfaces.

Under Linux 2.6, the LNET configuration parameters can be viewed under /sys/module/; generic and acceptor parameters under 'lnet' and LND-specific parameters under the corresponding LND's name.

Under Linux 2.4, sysfs is not available, but the LND-specific parameters are accessible via equivalent paths under /proc.

Notes about quotes: Depending on the Linux distribution, options with included commas may need to be escaped by using single and/or double quotes. Worst-case quotes look like this:

```
options lnet 'networks="tcp0,elan0"' 'routes="tcp
[2,10]@elan0"'
```

But the additional quotes may confuse some distributions. Check for messages such as:

```
lnet: Unknown parameter `'networks'
```

After modprobe LNET, the additional single quotes should be removed from modprobe.conf in this case.

Additionally, the message "refusing connection - no matching NID" generally points to an error in the LNET module configuration.

Note: By default, Lustre will ignore the loopback (lo0) interface. Lustre will not ignore IP addresses aliased to the loopback. Specify all Lustre networks in this case.

Liblustre network parameters may be set by exporting the environment variables LNET_NETWORKS, LNET_IP2NETS and LNET_ROUTES. Each of these variables uses the same parameters as the corresponding modprobe option.

Please note that it is very important that a liblustre client includes ALL the routers in its setting of LNET_ROUTES. A liblustre client cannot accept connections, it can only create connections. If a server sends RPC replies via a router that the liblustre client hasn't already connected to, these RPC replies will be lost.

> NOTE: Liblustre is not for general use. It was created to work with specific hardware (Cray) and should never be used with other hardware.

SilverStorm InfiniBand Options -

For the SilverStorm/Infinicon Infiniband LND (iiblnd), the network and HCA may be specificied, as in the example below:

options lnet networks="iib3(2)"

This says that this node is on iib network number 3, using HCA[2] == ib3.

3.3 Starting and Stopping LNET

LNET is started and stopped automatically by Lustre, but can also be started manually in a standalone manner. This is particularly useful to verify that your networking setup is working correctly before you attempt to start Lustre.

A few of the advantages of using LNET are explained below -

A. Performance

LNET routing can fully exploit all the performance optimizations available to the native interconnect between any pair of nodes. In other words, it can achieve full bandwidth of data transfer with minimum latency and CPU overhead.

It is not possible to have RDMA with TCP/IP. Thus, the endpoints of a Lustre connection (i.e. the client and the server) are forced to copy data. The zero-copy-tcp patch from CFS allows you to avoid this on the output. TCP/IP may also suffer from further unavoidable performance degradation when it is layered over another cluster interconnect. For example, IP over IB forces the use of UD (Unreliable Datagram) queue pairs which imposes a reduced MTU (Maximum Transmission Unit) and simply cannot achieve the same bandwidth as RC (Reliable Connected) queue pairs.

B. Load sharing

LNET automatically load balances over equivalent routers. It is achieved by monitoring message queue depth and queuing messages to the router with the shortest queue.

TCP/IP load balancing takes place with LNET configuration. Managing it may therefore vary depending on the available IP networks, hardware and management interfaces.

C. Resilience

Lustre monitors all its communications, and initiates retry and recovery algorithms when any of them fails. This allows LNET to abandon communications in case of failure, but at a cost. LNET delivers messages reliably in absence of failures in the underlying fabric. But when such failures occur, LNET flashes messages of being affected by the failure and does not retry. This is true irrespective of the network driver(s) in the message path.

LNET routing supports the marking of routers up or down on individual nodes. System administrators can fail out a router by marking it down on all the nodes, and bring it back into service again by marking it up on all the nodes. LNET automatically marks a router down locally, if communications with it complete with an error.

> NOTE: Currently this is implemented only for ELAN and TCP LNET networks. For all the remaining network types, it will be implemented as early as possible.

TCP achieves flexibility by retransmitting packets until safe receipt has been confirmed. Availability of alternative routes allows the effect of a failing IP router to be masked until IP route tables have been updated to route round it. If a site uses static IP routes, a failed IP router will reduce performance until it recovers. To prevent this, the site must run an automatic IP routing protocol over all IP networks, including the cluster "interior" network. Successful delivery must occur before the lustre timeout expires to avoid entering lustre in the retry/ recovery mode. TCP does not recognize the failed router, it just infers that its peer has not yet received some data it sent. So the second copy of the packet reaches there without running into the same problem. TCP requires redundant, load-balanced IP routes to work, and an IP routing protocol (e.g. RIP) to limit performance impact in the presence of failures.

The performance advantage of LNET routing is that it is built-in. It exploits the best drivers for the network available at all points. Obviously this advantage is unlikely to diminish with time.

3.3.1 Starting LNET

The command to start the lnet is -

```
$ modeprob lnet

$ lctl network up
```

To see the list of local nids -

```
$ lctl list_nids
```

This will tell you if your local node's networks are set up correctly. If not, see modules.conf "networks=" line and insure the network layer modules are correctly installed and configured.

To get the best remote nid -

```
$ lctl which_nid
```

This will take the "best" nid from a list of the nids of a remote host. The "best" nid is the one the local node will use when trying to communicate with the remote node.

3.3.1.1 Starting Clients

TCP client:

```
mount -t lustre mdsnode:/mdsA/client /mnt/lustre/
```

Elan client:

```
mount -t lustre 2@elan0:/mdsA/client /mnt/lustre
```

3.3.2 Stopping LNET

Before the LNET modules can be removed, LNET references must be removed. In general, these references are removed automatically during Lustre shutdown, but for standalone routers, an explicit step is necessary. It is to stop the LNET network by using the following command:

```
lctl network unconfigure
```

NOTE: Attempting to remove the Lustre modules prior to stopping the network may result in a crash, or an LNET hang. If this occurs, the node must be rebooted in most cases. So it is advised to be certain that the Lustre network and Lustre are stopped prior to module unloading, and to be extremely careful when using 'rmmod -f'.

To unconfigure LCTL network, following command can be used:

```
modprobe -r <any lnd and the lnet modules>

lconf --cleanup
```

This command will do the Lustre and LNET cleanup automatically in cases where lconf was used to start the services.

TIP:
To remove all the Lustre modules:
$ lctl modules | awk '{print $2}' | xargs rmmod

Chapter II – 4. Configuring Lustre - Examples

4.1 Simple TCP Network

Because the default network is tcp0, we omit the "@tcp0" everywhere. We can also use actual hostnames rather than numerical IP addresses within the lmc and mount commands. (NB hostnames, which are symbolic IP addresses can *not* be used in LNET module parameters.)

```
${LMC} --add net --node megan --nettype lnet  --nid
megan

${LMC} --add net --node oscar --nettype lnet  --nid
oscar

${LMC} --add net --node client --nettype lnet --nid
'*'
```

Modprobe.conf is the same on all nodes (we can omit this step as well, since this is the default) :

```
options lnet networks=tcp0
```

The servers megan and oscar are started with lconf, while clients are started by using the mount command:

```
mount -t lustre megan:/mdsA/client /mnt/lustre
```

4.2 Example 1: Simple Lustre Network

4.2.1 Installation Summary

- 8 OSS
- 1 MDS
- 40 Lustre Clients
- All lustre machines are dual processor Intel machines
- Storage is provided via DDN nodes
- All gigabit network

4.2.2 Usage Summary

- Typical usage of 17 clients and 2 OSS for continuous Lustre testing

  Testing new kernels and lustre versions
- Testing various configurations and software for integration into production clusters

4.2.3 Configuration Generation and Application

- Shell script runs ``lmc'' to generate XML
- Lustre XML stored in shared NFS
- Zeroconf mounting used
- Configurations are also generated using the ltest framework

  Uses same structure of shell script and shared NFS, but the hostnames and devices are determined by the test names.

4.3 Example 2: Lustre with NFS

4.3.1 Installation Summary

- 5 OSS
- 1 MDS
- 1 Lustre Client and NFS Server
- 1 AIX NFS Client
- All Lustre machines are dual processor Intel machines
- Storage is provided from serial ATA drives connected to 3ware RAID cards with 32TB total available
- Two gigabyte Ethernet networks, one each for Lustre and NFS
- Streaming I/O performance over NFS is typically 50-75% of Lustre
- No failover enabled or configured
- Initially 3 OSS configured, 2 more added when data from GFS was migrated in

4.3.2 Usage Summary

- Near-line storage for ESMF computing facility
- Replaced GFS – very poor performance and reliability, could not get passed 2TB/fs limit
- Usage is for scientific application data that is pushed off the local AIX storage

4.3.3 Configuration Generation and Application

- Shell script runs “lmc” to generate XML
- Scp used to distribute XML to Lustre nodes
- Custom script used from AIX client to start Lustre, NFS server and mount NFS client
- Zeroconf not used, could be added without issue
- Need to verify it does not reorder LOV when additional OSTs are added

4.4 Example 3: Exporting Lustre with Samba

4.4.1 Installation Summary

- 2 OSS
- 1 MDS
- 2 Lustre Clients; on exports Lustre via Samba
- 1 Windows Samba client, 2 Mac samba clients
- Most Lustre machines are althons
- One flat gigabit Ethernet network
- Failover is enabled, no failover pairs configured

4.4.2 Usage Summary

- /home filespace for Linux
- My Documents stored from Windows
- Both Mac and Windows access streaming music stored in Lustre
- Streaming I/O of 70-80M/s, ~18M/s over samba

4.4.3 Model of Storage

Typical Webfarm Home Network

4.4.4 Configuration Generation and Application

- Shell script runs ``lmc'' to generate XML
- Lconf used to start MDS/OST by hand
- Zeroconf mounting used on Lustre clients

4.5 Example 4: Heterogeneous Network with Failover Support

4.5.1 Installation Summary

- 64 OSS, 96 OSTs

  Each OSS has 2 gigE connections
- 2 MDS, 1 primary but not configured for failover
- 16 Portals Routers

  Each has 4 gigE and 1 Elan connection
- ~1000 Lustre Clients

  Once federated gigE network is installed hope to mount another 2000 clients (MCR clients and BGL IONs)
- Heterogeneous:

  Ia64 clients, MDS, Portals routers

  Ia32 OSS
- Storage is provided with 12 tiers of disk over 8 DDN nodes
- Failover is enabled, no failover pairs configured

  Hardware is there, but not configured to to perceived/real lack of reliable failover software and its ease of integration with Lustre

4.5.2 Usage Summary

- Scientific computation
- ~180 TB File system
- Theoretical best performance: 6.4GB/s;

  Best observed: 5.5GB/s;

  Typical observed: 2.0GB/s

4.5.3 Model of Storage

Typical LARGE HPC installation with mixed networking

4.5.4 Configuration Generation and Application

- Home grown bash scripts provide batch input to LMC
- XML is still edited by hand when new configurations are being tested

  Rare problems with the python XML parser
- Configuration files distributed by custom configuration mgmt
- Lctl used in home grown scripts to check certain values against known correct values to verify health of servers or clients
- /etc/init.d/lustre used to start OSTs, Routers, and MDS
- zconf used whenever possible -- much faster and ``nicer''

4.6 Example 5: OSS with Multiple OSTs

4.6.1 Installation Summary (*target)

- 224 OSS, 448 OSTs
  Each OSS has 2 gigE connections
- 2 MDS, 1 primary but not configured for failover
- ~1024 Lustre Clients -- I/O Nodes (ION)
- 64 Compute nodes (CN) per ION
  Total of 65,536 CN
  CN do not see Lustre, I/O forwarded to IONs
- All gigabit Ethernet networking for Lustre
  Compute nodes communicate with ION through Tree Network
- Storage is provided with 16 tiers on 8 DDN nodes
- In initial stages of Lustre testing now

4.6.2 Usage Summary

- Scientific computation
- ~900 TB Filesystem
- Theoretical best performance: 40GB/s

4.6.3 Model of Storage

Next generation ultra-large HPC installation

4.6.4 Configuration Generation and Application

The configuration generation and application of 'Example 4: OSS with Multiple OSTs' is quite similar to that of 'Example 3: Heterogeneous Network with Failover Support'. Please refer 2.5.4.

4.7 Example 6: Client with Sub-clustering Support

4.7.1 Installation Summary

- 104 OSS
- 1 MDS
- 1280 Lustre Clients

  Clients are arranged in sub-clusters of 256 nodes
- All lustre machines are dual processor Intel machines
- Storage is provided via DDN nodes
- All gigabit network
- Peak performance of 11.1GB/s

4.7.2 Usage Summary

- NSF grants allocations for researchers all over the world
- General scientific load Chemistry, cosmology, weather

4.7.3 Configuration Generation and Application

- Shell script runs ``lmc'' to generate XML
- Lustre XML stored in shared NFS
- Zeroconf mounting used

Chapter II – 5. More Complicated Configurations

5.1 Multihomed servers

Servers megan and oscar each have 3 tcp NICs (eth0, eth1, and eth2) and an elan NIC. eth2 is used for management purposes and should not be used by LNET. TCP clients have a single TCP interface, and Elan clients have a single Elan interface.

5.1.1 Modprobe.conf

Options to modprobe.conf are used to specify the networks available to a node. Two different options can be used; the *networks* option explicitly lists the networks available, and the *ip2nets* option provides a list-matching lookup. Only one of these options should be used. The order of LNET lines in modprobe.conf is important. If a node can be reached using more than one network, the first one specified in modprobe.conf will be chosen.

Networks

On the servers:

```
options lnet 'networks="tcp0(eth0,eth1),elan0"'
```

Elan-only clients:

```
options lnet networks=elan0
```

TCP-only clients:

```
options lnet networks=tcp0
```

NOTE: In case of TCP-only clients, all the available IP interfaces will be used for tcp0 since the interfaces are not specified. If there were more than one, the IP of the first one found is used to construct the tcp0 NID.

ip2nets

The *ip2nets* option would typically be used to provide a single, universal modprobe.conf file that can be used on all servers and clients. An individual node identifies the locally available networks based on the listed IP address patterns that match the node's local IP addresses. Note that the IP address patterns listed in this option (ip2nets) are used **only** to identify the networks that an individual node should instantiate. They are **not** used by LNET for any other communications purpose. The servers megan and oscar have eth0 ip addresses 192.168.0.2 and .4. They also have IP over Elan (eip) addresses of 132.6.1.2 and .4. TCP clients have ip addresses 192.168.0.5-255. Elan clients have eip addresses of 132.6.[2-3].2, .4, .6, .8.

Modprobe.conf is identical on all nodes:

```
options lnet 'ip2nets=" \
tcp0(eth0,eth1)192.168.0.[2,4]; \
```

```
tcp0 192.168.0.*; \
elan0 132.6.[1-3].[2-8/2]"'
```

NOTE: Lnet lines in modprobe.conf are used by the local node only to determine what to call its interfaces. They are not used for routing decisions.

Because megan and oscar match the first rule, LNET will use eth0 and eth1 for tcp0 on those machines; although they also match the second rule, it is the first matching rule for a particular network that will be used. The servers also match the (only) elan rule. The [2-8/2] format matches the range 2-8 stepping by 2; i.e. 2,4,6,8. Clients at e.g. 132.6.3.5 would not find a matching elan network.

5.1.2 LMC configuration preparation

The tcp NIDs specified should use the address of the first TCP interface listed in the *networks* or *ip2nets* options line above (eth0).

```
${LMC} --add net --node megan --nettype lnet --nid
192.168.0.2@tcp0

${LMC} --add net --node megan --nettype lnet --nid
2@elan

${LMC} --add net --node oscar --nettype lnet --nid
192.168.0.4@tcp0

${LMC} --add net --node oscar --nettype lnet --nid
4@elan

A single client profile will work for both tcp and
elan clients:

${LMC} --add net --node client --nettype lnet --nid
'*'
```

NOTE: The example above shows that in --add net option for each interface the --node parameter is the same but the --nid parameter is changing, which specifies the NID of the interface.

5.1.3 Start servers

Start servers with lconf. The recommended order is the OSSs then the MDS lconf config.xml.

5.1.4 Start clients

Tcp clients can use the hostname or ip address of the MDS:

```
mount -t lustre megan@tcp0:/mdsA/client /mnt/lustre
```

Elan clients will be started with:

```
mount -t lustre 2@elan0:/mdsA/client /mnt/lustre
```

5.2 Elan to TCP routing

Servers megan and oscar are on the elan network with eip addresses 132.6.1.2 and .4. Megan is also on the TCP network at 192.168.0.2 and routes between TCP and elan. There is also a standalone router router1 at elan 132.6.1.10 and tcp 192.168.0.10. Clients are on either elan or tcp.

5.2.1 Modprobe.conf

Modprobe.conf is identical on all nodes:

```
options lnet 'ip2nets="tcp0 192.168.0.*; \
elan0 132.6.1.*"' 'routes="tcp [2,10]@elan0; \
elan 192.168.0.[2,10]@tcp0"'
```

5.2.2 LMC configuration preparation

```
${LMC} --add net --node megan --nettype lnet --nid
192.168.0.2@tcp0
${LMC} --add net --node megan  --nettype lnet  --nid
2@elan
${LMC} --add net --node oscar --nettype lnet --nid
4@elan
${LMC} --add net --node client --nettype lnet  --nid
'*'
```

5.2.3 Start servers

router1

```
modprobe lnet
lctl network configure
```

megan and oscar:

```
lconf route.xml
```

5.2.4 Start clients

tcp client:

```
mount -t lustre megan:/mdsA/client /mnt/lustre/
```

elan client:

```
mount -t lustre 2@elan0:/mdsA/client /mnt/lustre
```

Chapter II – 6. Failover

6.1 What is Failover?

We say a computer system is 'Highly Available' when the services it provides are available with minimum downtime. Even in case of failure conditions such as loss of a server, or network or software fault, the services being provided remain unaffected for the user. This is because any services on a highly available system are transparent to clients, or cause minimum disruption. We generally measure availability by the percentage of time we require the system to be available.

Failover is a service that automatically switches to a standby server when the primary system fails or the service is temporarily shut down for maintenance. It is an important fault tolerance function of mission-critical systems that rely on constant accessibility. Failover is automatic and completely application-transparent supporting both the Meta-data and the Object servers. This availability is almost always maintained by providing replicated hardware and/or software, so that failure of any system will be covered by a paired system.

The Lustre failover requires two nodes (a failover pair), which must be connected to a shared storage device.

Lustre provides a file system resource. To keep the file system highly availble, Lustre supports failover at the server level. It requires a third party tool to support failover. The recommended choice is the Heartbeat package from linux-ha.org. Lustre will work with any HA software that supports resource (I/O) fencing. The Heartbeat software is responsible for detecting failure of the primary server node and controlling the failover. This requires a pair of servers with a shared connection to a physical storage (Like SAN, NAS, hardware raid, SCSI, Fider Channel). The method of sharing the storage is essentially transparent to Lustre. To ensure high availability at the level of physical storage, we encourage the use of RAID arrays to protect against drive-level failures.

To have a fully automated high available Lustre system, one needs a power management software and HA software, which must provide the following -

A) -- Resource fencing - The physical storage must be protected from simultaneous access by two nodes

B) -- Resource control - Starting and stopping the Lustre processes as part of failover, maintaining cluster state, etc.

C) -- Health monitoring - Verifying the availability of hardware and network resources, responding to health indications given by Lustre.

For proper resource fencing, the Heartbeat software must be able to completely power off the server or disconnect it from the shared storage device. It is absolutely vital that no two active nodes access the same partition, at the risk of severely corrupting data. When the Heartbeat detects a server failure, it calls a process (STONITH) to power off the node; and then starts Lustre on the secondary node. The HA software can call /etc/init.d/lustre for the said purpose.

The servers are arranged in pairs for the purpose of failover. A system administrator can failover manually, with lconf. When an lconf --cleanup --failover command is issued, the disk device is set read-only. This allows the second node to start service using that same disk, after the command completes. This is known as a "soft" failover, in which case both the servers can be running and connected to the net. Powering the node off is known as a "hard" failover.

To automate failover with Lustre, one needs a power management software, remote control power equipment, and HA software.

6.1.1 The Power Management Software

The linux-ha package includes a set of power management tools, known as STONITH (Shoot The Other Node In The Head). STONITH has native support for many power control devices, and is extensible. PowerMan, by the Lawrence Livermore National Laboratory, is another tool for manipulating remote power control (RPC) devices from a central location. Several RPC varieties are supported natively by PowerMan.

The latest version is available on

http://www.llnl.gov/linux/powerman/

6.1.2 Power Equipment

A multi-port, Ethernet addressable Remote Power Control is relatively inexpensive. Consult the list of supported hardware on the PowerMan site for recommended products. Linux Network Iceboxes are also very good tools. They combine both the remote power control and the remote serial console into a single unit.

6.1.3 Heartbeat

The heartbeat program is one of the core components of the Linux-HA (High-Availability Linux) project. Heartbeat is highly portable, and runs on every known Linux platform, and also on FreeBSD and Solaris.

For more information, see:

http://linux-ha.org/heartbeat/

For download, go to:

http://linux-ha.org/download

Lustre is very connection-oriented; not so much low-level connections like TCP, but rather high-level. Lustre connections are a strong concept in the architecture.

Lustre can be in one of the following three states (effectively):

Connected: If a connection is "active", then the client has done a handshake with the server and the connection is considered to be alive. New requests sent for this connection will go to sleep until either a reply arrives or the connection becomes disconnected or failed. If there is no traffic on a given connection, the client will periodically check the connection to ensure that it is still connected to the server.

Disconnected: If a connection is "disconnected", then at least one request has timed out. New and old requests alike are put to sleep until one of the following takes place -

- A reply arrives (which would only happen after the connection is activated and the requests asynchronously re-sent) ;
- The application gets a signal (such as TERM or KILL);
- The client is evicted by the server (which would return-EIO for these requests); or
- The connection becomes "failed".

In other words, the timeout is effectively infinite. Lustre will wait as long as it needs to, to avoid giving the application an -EIO error (which most would not know how to handle).

Failed: Finally, if a connection is "failed", which is what happens immediately in the "failout" OST mode, the new and old requests receive -EIO immediately. Normally (in the non-failout mode), a connection can get into this state only by using "lctl deactivate", which would only be done on the client in the event of an OST failure that an administrator doesn't think can be quickly corrected.

Types of Nodes in a Failover: To configure a Failover, 2 nodes are needed. One can configure these nodes in 2 ways – active/active and active/passive. An 'active' node actively serves data and a 'passive' node is idle, standing by to take over in the event of a failure. In the example case of using 2 OSTs (both of which are attached to the same shared disk device), the following failover configurations are possible:

'active/ passive' - This configuration has two nodes out of which only one is actively serving data all the time. In case of a failure, the other node takes over.

If the active node fails, the OST in use by the active node will be taken over by the passive node which is now active. This node will serve most of the services that were on the failed node.

'active/ active' - This configuration has two nodes actively serving data all the time. In case of a failure, one node would take over for the other.

To configure this with respect to the shared disk, the shared disk would need to be split into 2 partitions, and each of the OSTs would be the 'primary' server for one partition and the 'secondary' server for the other partition.

6.2 OST Failover Review

The OST has two operating modes: failover and failout. The default mode is failover. In this mode, the clients reconnect after a failure, and the in-progress transactions get completed. Data on the OST is written synchronously, and the client replays uncommitted transactions after the failure.

In failout mode when any communication error occurs, the client attempts to reconnect, but is unable to continue with the transactions that were in progress during the failure. Also, if the OST actually fails, data that has not been written to the disk (still cached on the client) is lost. Applications usually see an -EIO for operations done on that OST until the connection is reestablished. However, the LOV layer on the client avoids using that OST, so the operations such as file creates and fsstat still succeed.

6.3 MDS Failover Review

The MDS has only one failover mode: active/passive. Even with only one MDS node configured, the clients attempt to recover transparently to the applications, and replay the uncommitted transactions.

6.4 Failover Configuration

The failover MDS and OSTs are configured in essentially the same way – multiple objects are added to the configuration with the same service name. The --failover option is specified on at least one of the objects to enable the failover mode. (This is required to enable failover on the OSTs.) For example, to create a failover OST named ost1 on nodes nodeA and nodeB with a shared disk device referenced on both nodes as /dev/sdb1; the steps that can be followed are as below:

```
lmc --add ost --ost ost1 --failover --node nodeA --
lov lov1 --dev /dev/sdb1

lmc --add ost --ost ost1 --failover --node nodeB --
lov lov1 --dev /dev/sdb1
```

6.4.1 Active Server Management with XML configurations

The current active server can be maintained by including either a simple HTTP server and wget, or LDAP, or shared storage. When static XML config files are being used, the active node is usually stored in separate files on a shared file system, such as NFS.

6.4.2 Shared Resource

The lconf --select option is used to override the current active node for a particular service.

For example, one way to manage the current active node is to save the node name in a shared location that is accessible to the client upcall. When the upcall runs, it determines which service has failed, and looks up for the current active node to make the file system available. The current node and the upcall parameters are then passed to lconf in order to complete the recovery.

Using the above example, consider that nodeA fails. In this case, when nodeB starts the ost1 service, it needs to save "nodeB" in a file named ost1_UUID. Then it starts the service with the following command:

```
lconf --node nodeB --select ost1=nodeB <config.xml>
```

When the clients detect a connection failure and call the upcall, the upcall script will get the parameters passed as follows:

argv[1] = "FAILED_IMPORT"

argv[2] = target uuid

argv[3] = obd name

argv[4] = remote connection uuid

argv[5] = obd uuid

For example:

```
$upcall FAILED_IMPORT ost1_UUID
OSC_localhost_ost4_MNT_localhost NET_uml2_UUID
ff151_lov1_6d1fce3b45

lconf --recover --select ost1=nodeB --target_uuid $2
--client_uuid $3 --conn_uuid $4  <config.xml>
```

6.4.3 Active/Active Failover Configuration

OST failover is typically configured as an active/active configuration where two servers are paired a such a way that each is primary for one storage device and secondary for another. With current Lustre there can be only one active MDS so MDS failover requires an active/passive server pair. If the shared storage used for a failover OST is configured with multiple LUNs, then it is possible to have a load balanced active/active configuration. Each node is the primary node for a group of OSTs, and the failover node for other groups. To expand the simple two-node example, we add ost2 which is primary on nodeB, and is on the LUNs nodeB:/dev/sdc1 and nodeA:/dev/sdd1.

```
lmc --add ost --ost ost1 --failover --node nodeA --
group nodeA --lov lov1 --dev /dev/sda1

lmc --add ost --ost ost1 --failover --node nodeB --
lov lov1 --dev /dev/sdb1

lmc --add ost --ost ost2 --failover --node nodeB --
group nodeB --lov lov1 --dev /dev/sdc1

lmc --add ost --ost ost2 --failover --node nodeA --
lov lov1 --dev /dev/sdd1
```

If the --group nodeB option is used, then only the active services in group nodeB will be started. This is generally used on a node, which is already running the services. If it is not used, then all the services active on nodeA will be started, which is generally what is needed at the boot time.

To return to the original load-balanced configuration, first the service is stopped in the failover mode. It is restarted on the original node, and then the active node is updated for the affected services. The clients will treat this as a failover, and recover normally.

on nodeA, limit the scope to only group nodeB

```
lconf --node nodeA --group nodeB --cleanup --force --
failover --select ost2=nodeA <config.xml>
```

on nodeB

```
lconf --node nodeB [--group nodeB] --select
ost2=nodeB <config.xml>
```

If the --group nodeB option is not used in the first command, then all the services on nodeA will be stopped forcibly.

6.4.4 Configuration

Prerequisites –

Hardware

Shared Storage

Multiple inter connections betweens node's health checking, preferably over different media types, typically one Ethernet and one serial connection

Software

heartbeat-pils-1.2.3 OR heartbeat-pils-2.0.4-1

heartbeat-stonith-1.2.3 OR heartbeat-stonith-2.0.4-1

heartbeat-1.2.3 OR heartbeat-2.0.4-1

Tools will be installed in /usr/lib/heartbeat or /usr/lib64/heartbeat on x86_64 platforms. The tool set in version 2.x is much richer than that in the version 1.x and also includes tools required to migrate a 1.x configuration to 2.x.

Installations – Install these packages using standard RPM techniques.

After installation, verify that heartbeat tools are installed in either /usr/lib/heartbeat or /lib/heartbeat; and /etc/ha.d have been created.

Install Mon - Perl

Mon is a Perl package that we use for monitoring of Lustre status.

The base package is available from

ftp.kernel.org:/pub/software/admin/mon

Mon requires these Perl packages: (Install from CPAN if possible)

Time::Period

Time::HiRes

Convert::BER

Mon::SNMP

For CPAN, do:

```
# perl -MCPAN -eshell
```

If you have never used CPAN, go through the manual configuration,

the defaults are correct in most of the cases. After setup completes,

> install Time::Period Time::HiRes Convert::BER Mon::SNMP

and answer the prompts if any.

For standard Perl tarballs:

- untar

- cd into the resulting directory.

```
# perl Makefile.PL
# make
# make install
```

Main Mon installation

After installing the Perl packages, get the mon tarball from

ftp.kernel.org:/pub/software/mon

- Untar
- Copy the 'mon' binary to a location in the root's PATH (/usr/lib/mon/mon is default)
- Create a set of mon directories, specified in /etc/mon/mon.cf

cfbasedir = /etc/mon

alertdir = /usr/local/lib/mon/alert.d

mondir = /usr/local/lib/mon/mon.d

statedir = /usr/local/lib/mon/state.d

logdir = /usr/local/lib/mon/log.d

dtlogfile = /usr/local/lib/mon/log.d/downtime.log

- Copy 'lustre_health.monitor' into $mondir.
- Copy 'lustre_fail.alert into $alertdir.
- Copy 'mon_trap' into a convenient location.
- Copy 'mon' file (supplied) or S99mon to /etc/ha.d/resource.d/mon

6.4.4.1. Hardware

- The setup must consist of a failover pair where each node of the pair has access to the shared storage. If possible, the storage paths should be identical (nodeA:/dev/sda == nodeB:/dev/sda)
- The shared storage can be arranged in an active/passive (MDS, OSS) or active/active (OSS only) configuration. Each shared resource should have a primary (default) node. The secondary node is assumed.
- Both the nodes must have one or more communication paths for the heartbeat traffic. The communication path can be:

  - a dedicated Ethernet
  - a serial live (serial crossover cable)

Failure of all the heartbeat communication is not good. This condition is called 'split-brain', and the heartbeat software resolves this situation by powering down one of the nodes.

The heartbeat software provides a remote ping service, which is used to monitor the health of the external network (ipfail). If you wish to use this service, you must have a very reliable external address to use as the ping target.

6.4.4.2 Lustre Configuration

- Create the directory '/etc/lustre'
- Copy the XML file to '/etc/lustre/config.xml'
- Verify that /etc/init.d/lustre exists
- Note the names of the OST and MDS resources
- Decide which node 'owns' each resource

6.4.4.3 Heartbeat Configuration

As mentioned above, if you have a Heartbeat 1.x. configuration, you can run with that. To convert from version 1.x. Configuration, use the 'haresources2cib.py' script, typically found in /usr/lib/heartbeat. If you want to install heartbeat 2.x, you are recommended to create a v1-style configuration and convert, as the version 1 files are human-readable. The heartbeat XML configuration lives at /var/lib/heartbeat/cib.xml and the new resource manager is enabled with the 'crm yes' directive in /etc/ha.d/ha.cf.

Heartbeat log daemon

Heartbeat 2.x adds a logging daemon which manages logging on behalf of cluster clients. The UNIX syslog API calls may get blocked. This daemon prevents a busy syslog from triggering a false failover. The logging configuration has been moved to /etc/logd.cf - the directives are essentially unchanged.

> NOTES: Heartbeat v2 will run without altering the version 1.x configuration files. This makes upgrading simple. You can test the basic function and quickly roll back if issues appear. Heartbeat v2 does not require a virtual IP address to be associated with a resource. Since we don't use virtual IPs, this is very useful.

1. Basic Configuration - no STONITH.

It is good to test with this configuration before adding STONITH.

Assuming two nodes, 'nodeA' and 'nodeB'

- nodeA owns 'ost1'
- nodeB owns 'ost2'
- dedicated ethernet – 'eth0'
- serial crossover link - '/dev/ttyS1'
- remote host for health ping - '192.168.0.3

a. Create /etc/ha.d/ha.cf

- Should be identical on both the nodes
- The order of directives matters

b. Create /etc/ha.d/haresources

- This file must be identical on both the nodes
- It specifies a virtual IP address, and a service
- The virtual IP address should be a subnet matching a physical Ethernet. Failure to do so results in error messages, but is not fatal. For heartbeat version 2.x, haresources no longer requires the dummy virtual IP address. The basic.ha.cf wil look some what like -

```
use_logd on

keepalive

deadtime

initdead

udpport

bcast eth1

baud 19200

serial /dev/ttyS1

auto_failback off

crm yes

node nodeA nodeB

ping 192.168.0.60

respawn hacluster /usr/lib/heartbeat/ipfail
```

```
# /usr/lib/heartbeat/haresources2cib.py -c
basic.ha.cf    basic.haresources > basic.cib.xml
```

c. Create /etc/ha.d/authkeys -

Copy example from /usr/share/doc/heartbeat-<version>

d. Create symlinks from /etc/init.d/lustre to

/etc/ha.d/resource.d/<lustre service name>

```
# ln -s /etc/init.d/lustre/etc/ha.d/  \

resource.d/ost1

# ln -s/etc/init.d/lustre/etc/ha.d/  \

resource.d/ost2
```

e. The contents of the cib.xml file will look as below.

A. The first section in the XML file is attributes

<attributes>

<nvpair id="**symmetric_cluster**" name="**symmetric_cluster**" value="**true**" />

<nvpair id="**no_quorum_policy**" name="**no_quorum_policy**" value="**stop**" />

CHANGE TO

<nvpair id="**no_quorum_policy**" name="**no_quorum_policy**" value="**ignore**" />

<nvpair id="**default_resource_stickiness**" name="**default_resource_stickiness**" value="**0**" />

<nvpair id="**stonith_enabled**" name="**stonith_enabled**" value="**false**" />

<nvpair id="**stop_orphan_resources**" name="**stop_orphan_resources**" value="**false**" />

<nvpair id="**stop_orphan_actions**" name="**stop_orphan_actions**" value="**true**" />

<nvpair id="**remove_after_stop**" name="**remove_after_stop**" value="**false**" />

<nvpair id="**short_resource_names**" name="**short_resource_names**" value="**true**" />

<nvpair id="**transition_idle_timeout**" name="**transition_idle_timeout**" value="**5min**" />

<nvpair id="**is_managed_default**" name="**is_managed_default**" value="**true**" />

</attributes>

The default values should be fine for most installations.

B. The actual resources are defined in the primitive section

<primitive class="**lsb**" id="**ost1_1**" type="ost1" resource_stickiness="INFINITY">>

<operations>

<op id=**"ost1_mon"** interval=**"120s"** name=**"monitor"** timeout=**"60s"**/>

CHANGE TO

<op id=**"1"** interval=**"10s"** name=**"monitor"** timeout=**"30s"**/>

</operations>

</primitive>

<primitive class="**lsb**" id="**ost2_2**" provider="**heartbeat**" type="**ost2**">

<operations>

<op id="**ost2_mon**" interval="**120s**" name="**monitor**" timeout="**60s**" />

<op id=**"2"** interval=**"10s"** name=**"monitor"** timeout=**"30s"**/>

</operations>

</primitive>

The default behavior of Heartbeat is automatic failback of resources when a server is restored. To avoid this, we must add a parameter to the <primitive> definition. We would also like to tighten up the timeouts a bit. In addition, the current version of the script does not name parameters correctly.

- Copy the modified resource file to /var/lib/heartbeat/crm/cib.xml
- Start heartbeat
- After startup, Heartbeat will re-write the cib.xml, adding a <node> section and status information. Do not alter those fields.

> NOTE: Heartbeat v2 supports multi-node (>2 node) clusters (not tested on more than 2 nodes). The multi-node setup adds a 'score' value to the resource configuration which is used to decide the proper node for a resource when failover occurs.

2. Adding STONITH

The two nodes must have a method to control each other's state. Hardware remote power control is best. There must be a script to start and stop a given node from the other node. STONITH provides 'soft' power control methods (ssh, meatware) but these cannot be used in a production situation.

Much must be deduced by running the stonith package by hand. Stonith has some supplied packages, but can also run with an external script.

-- There are two Stonith modes

a. Single STONITH command for all nodes:

/etc/ha.d/ha.cf

stonith <type> <config file>

b. The STONITH command per-node:

/etc/ha.d/ha.cf

$ stonith_host <hostfrom> <stonith_type> <params...>

Using an external script to kill each node

$stonith_host nodeA external foo /etc/ha.d/reset-nodeB

stonith_host nodeB external foo /etc/ha.d/reset-nodeA

'foo' is a placeholder for an un-used parameter

To get the proper syntax:

$stonith -L

lists supported models

$stonith -l -t <model>

-- lists parameters required,and specify config file name.

-- You should attempt a test with

$ stonith -l -t <model> <fake host name>

-- This will also give data on what is required.

-- You should be able to test by using a real host name

-- external stonith scripts should take parameters

{start|stop|status}

and return 0 or 1

3. Mon (status monitor)

Mon requires two scripts

- a monitor script, which checks a resource for health
- an alert script, which is triggered by the failure of a monitor

Mon requires one configuration file

- /etc/mon/mon.cf
- There is one /proc/fs/health_check per node
- one monitor only per node with this design
- make sure /etc/ha.d/resource.d/mon exists
- add mon to haresources (see haresources.with.mon)
- see simple.health_check.monitor and

health_check.monitor

For example,

- Copy your monitor script to the mon.d directory
- Configure fail_lustre.alert to perform the actions you wish to perform on node failure
- Copy fail_lustre.alert to your mon alert.d directory
- Restart heartbeat

4. Operation

In normal operation, Lustre should be controlled by heartbeat. Heartbeat should be started at the boot time. It will start Lustre after the initial dead time.

A. Initial startup

- Stop heartbeat if running.
- If this is a new Lustre file system:

```
lconf --reformat /etc/lustre/config.xml
lconf --cleanup /etc/lustre.config.xml
```

- If this is a new configuration for an existing Lustre file system, remember to lconf write_conf on the MDS

```
$ lconf --write_conf /etc/lustre/config.xml
$ lconf --clenaup /etc/lustre/config.xml
```

To start the hearbeat on one node

```
$ /etc/init.d/heartbeat start
$ tail -f /var/log/ha-log to see progress
```

- After initdead, this node should start all the Lustre objects

```
$ /etc/init.d/heartbeat start on the second node
```

- After heartbeat is up on both the nodes, failback the resources to the second node. On the second node, run:

```
# /usr/lib/heartbeart/hb_takeover local
```

- You should see the resources stop on the first node, and start up on the second node.

B. Testing

-- Pull power from one node

-- Pull networking from one node

-- After mon is setup, pull connection between OST and backend storage.

C. Failback

-- In normal case, failback is done manually after the operator has determined that the failed node is now in a working state. The Lustre clients can be doing work during a failback, but will block momentarily.

> NOTE: Heartbeat v2 adds a 'resource manager' (crm). The resource configuration is maintained as an XML file. This file is re-written by the cluster frequently. Any alterations to the configuration should be made with the HA tools or when the cluster is stopped.

Chapter II – 7. Configuring Quotas

7.1 Working with Quotas

Quotas allow a system administrator to limit the maximum amount of disc space a user's (or group's) home directory can take up. (It can also be used on any other directory.) This prevents home directories from becoming too large. Quotas are set by root, on a per partition basis, and can be set for both, individual users and/or groups. Before a file is written to a partition where quotas have been set, the quota of the creator's group is checked first. If a quota for that group exists, the size of the file is counted towards that group's quota. If no quota exists for the group, the owner's user quota is checked before the file is written.

The idea behind quota is that users are forced to stay under their disk consumption limit, taking away their ability to consume unlimited disk space on a system. Quota is handled on per user per file system basis. If there is more than one file system which a user is expected to create files in, then quota must be set for each file system separately.

It is used most often on timesharing systems where it is desirable to limit the amount of resources any one user or group of users may allocate. This prevents one user or group of users from consuming all of the available space.

7.1.1 Configuring Disk Quotas

Enabling Quotas

As root, using the text editor of your choice, add the usrquota and/or grpquota options to the file systems that require quotas. Enable the quotas per file system by modifying /etc/fstab.

LABEL=/	/	ext3	defaults	1 1
LABEL=/boot	/boot	ext3	defaults	1 2
none	/dev/pts	devpts	gid=5,mode=620	0 0
mdsname:/mdtname/client	/mnt/lustre	ldiskfs	defaults,usrquota,grpquota	1 2
none	/proc	proc	defaults	0 0
none	/dev/shm	tmpfs	defaults	0 0
/dev/hda2	swap	swap	defaults	0 0

In this example, the /mnt/lustre file system has both, the user and the group quotas enabled.

NOTE: Lustre with Linux Kernel 2.4 will not support quotas.

7.1.2 Remounting the File Systems

After adding the userquota and grpquota options, remount each file system whose fstab entry has been modified. If the file system is not in use by any process, use the unmount command followed by the mount command to remount the file system. If the file system is currently in use you will not be able to unmount the file system. You can see what process and users are accessing the file system, and preventing you from unmounting the file system, with the following command -

```
$ fuser -mv /mnt/lustre
```

If you wish to kill all the processes that are accessing the file system you can use the command -

```
$ fuser -km /mnt/lustre
```

Exercise caution when using the "-km" switch as the users will not be warned before it kills their process. In some cases the easiest method for remounting the file system is to reboot the system.

7.1.3 Creating Quota Files

After each quota-enabled file system is remounted, the system is now capable of working with disk quotas. However, the file system itself is not yet ready to support quotas. The next step is to run the quotacheck command.

- quotacheck :- This Scans lustre file systems for disk usage and creates quota files. It is strongly recommended to run quotacheck on a static file system. Otherwise it may result in wrong statistic of disk usage in the quota files. Also, the quota will be turned on by default after quotacheck completes. The options that can be used are as follows:

- a — Check all the quota-enabled, locally-mounted file systems
- v — Display verbose status information as the quota check proceeds
- u — Check the user disk quota information
- g — Check the group disk quota information

```
$lfs quotacheck -ug /mnt/lustre
```

- quotaon :- Quotaon announces to the system that disk quotas should be enabled on one or more file systems. The file system quota files must be present in the root directory of the specified file system. The following command can be used with lmc (Lustre Configuration Maker). For more details on lmc, see Chapter V – 1. 'User Utilities (man1)'.

```
$ lmc --add node --node node --quota 'quotaon=ug'
```

  The above command can be used only for MDSs and OSTs.

- quotaoff :- Quotaoff command announces to the system that the specified file systems should have any of the disk quotas turned off.

- setquota :- Setquota is a command line quota editor. The file system, user/group name and new quotas for this file system can be specified on the command line.

```
$ lfs setquota -u bob 0 10000000 0 1000000
/mnt/lustre

$ lfs setquota -g tom 0 10000000 0 1000000
/mnt/lustre
```

  Description: Set limits for a user "bob" and a group "tom": Block hardlimit is around 10GB, and inode hardlimit 1M. At present, softlimits are not supported in Lustre, so set block softlimit and inode softlimit as 0. It is strongly recommended to run setquota on a non-busy lustre file system. Otherwise possible wrong block hardlimit value can be set.

- quotachown :- This sets or changes the file owner and the group on OSTs of the specified file system usage

```
$ lfs quotachown –I /mnt/lustre
```

Chapter II – 8. Upgrading From 1.4.5

8.1 Portals and LNET Interoperability

LNET uses the same wire protocols as portals, but has a different network addressing scheme; i.e. the portals and LNET NIDs are different. In single-network configurations, LNET can be configured to work with portals NIDs so that it can inter-operate with portals and can allow a live cluster to be upgraded piecemeal. This is controlled by the 'portals_compatibility' module parameter which is described below.

With Portals compatibility configured, old XML configuration files remain compatible with LNET. The 'lconf' configuration utility recognizes the portals NIDs and converts them to LNET NIDs.

Old client configuration logs are also compatible with LNET. Lustre running over LNET recognizes the portals NIDs and converts them to LNET NIDs, but issues a warning. Once all the nodes have been upgraded to LNET, these configuration logs can be rewritten to update them to LNET NIDs.

8.1.1 Portals Compatibility Parameter

The following module parameter controls interoperability with portals:

portals_compatibility="strong"|"weak"|"none"

"strong" is compatible with both, portals and with LNET running in either "strong" or "weak" compatibility mode. As this is the only mode compatible with portals, all the LNET nodes in the cluster must set this until the last portals node has been upgraded.

"weak" is not compatible with portals, but *is* compatible with LNET running in any mode.

"none" is not compatible with portals or with LNET running in "strong" compatibility mode.

8.1.2 Upgrade a cluster using shutdown

Upgrading a system that can be completely shut down is easier: Shutdown all the clients and servers, install an LNET release of Lustre everywhere, --write-conf the MDS, and restart everything. No portals_compatibility option is needed (the default value is "none").

When upgrading, you should install (rather than upgrade) the kernel and lustre-modules RPMs. This allows you to keep the older, known-good kernels installed in case the new kernel fails to boot.

First, upgrade the kernel using the following command -

```
$ rpm -ivh --oldpackage kernel-smp-2.6.9- \
22.0.2.EL_lustre \ .1.4.6.i686.rpm
```

Then, the upgrade the Lustre modules with the command -

```
$ rpm -ivh --oldpackage lustre-modules-lustre- \
modules-1.4.6- \
2.6.9_22.0.2.EL_lustre.1.4.6smp.i686.rpm
```

The lustre-modules RPM is kernel-specific, so if you have multiple kernels installed you will need a lustre-modules RPM for each kernel. We recommend using --oldpackage as sometimes RPM will report that an already installed RPM is newer, even though it may not be.

You can only have one lustre RPM (the userspace tools, not the lustre-modules RPM mentioned above) installed at a time, so you should upgrade this RPM with the command -

```
$ rpm -Uvh lustre-1.4.6- \
2.6.9_22.0.2.EL_lustre.1.4.6smp.i686.rpm
```

Before rebooting into the new Lustre kernel, double check your bootloader (grub, lilo) to make sure it will boot into the new kernel.

After Installing Packages -

After you install certain updates, you may need to take additional steps. For example, if you are upgrading to Lustre 1.4.6, you need to update your configuration logs.

Occasionally, it is necessary to update the configuration logs on the MDS.

Some examples of when this is needed are as follows:

- Enabling recovery on OSTs
- Changing the default striping pattern
- Changing the network address of a server
- Adding or removing servers
- Upgrading to a newer version

Follow the steps given below after installing the packages -

1. Shut down all the client and MDS nodes. This operation does not affect OSS nodes, so they do not need to be shut down at this time.

2. On the MDS node, run the following command:

```
$  lconf --write_conf /path/to/lustre.xml
```

3. Start OSS nodes if they were not already running.

4. Start the meta-data server as usual.

5. Mount Lustre on clients.

8.1.3 Upgrading a cluster "live"

A portals installation may be upgraded to LNET "live" in 3 phases as described below. To maximize service availability, servers (MDS and OSS) should be failed over to their backups while they are being upgraded and/or rebooted.

1 Shutdown the Lustre services on any node (servers or clients), using failover backups if desired for uninterrupted file system service. Remove old Lustre releases from the node. Upgrade the node to an LNET release of Lustre by installing the appropriate release RPMs. Configure the LNET options in modprobe.conf as appropriate for your network topology. [Note that only basic network topologies will be supported through the live upgrade process.] Set 'portals_compatibility="strong"' in the LNET modprobe.conf options. The Lustre services or client may now be restarted on this node. At this point, the node will be speaking "old" Portals, but will understand new LNET. This phase is only complete when *all* the nodes have been upgraded to LNET.

2 Ensure phase 1 is complete (i.e. all the nodes have been upgraded to LNET). Now set 'portals_compatibility="weak"' in the LNET modprobe.conf options on all the nodes. Nodes may now be rebooted (and servers failed over) in any order. As they are rebooted, the nodes will be speaking LNET, but will understand old Portals (which is still being spoken by the "stong" nodes). This phase is only complete when *all* the nodes are either down or running LNET in "weak" compatibility mode.

3 Ensure phase 2 is complete (i.e. all nodes are either down or are running LNET in "weak" compatibility mode). Now remove 'portals_compatibility' from the LNET modprobe.conf options on all the nodes (it defaults to "none"). Nodes may now be rebooted (and servers failed over) in any order. These nodes will now reject the old Portals protocol. This phase is only complete when *all* the nodes are either down or running LNET in the new mode.

Note that phase 3 must be complete before future upgrades are possible. Similarly, phase 3 must be complete before the site configuration can be updated to include multiple networks.

You may rewrite the client configuration logs after phase 1 has completed to avoid warnings about converting portals NIDs to LNET NIDs. As this requires an MDS service outage, you may choose to complete the upgrade in one step at this time by removing 'portals_compatibility' from the LNET modprobe.conf options on all the nodes and rebooting everywhere.

8.1.4 Upgrading from 1.4.5

For upgrading from 1.4.5 to 1.4.6, you need to download the latest RPM of Luster 1.4.6. If you want to upgrade a live Lustre, you need to failover to another server.

As 1.4.5 lustre modules have to be unloaded and then lustre 1.4.6 modules need to be loaded, a single node (Lustre witout failover) can not continue to run the lustre during the upgrde.

Following can be the steps for upgrading a live Lustre (Considering the node has a failover setup):

1. Download the latest RPM
2. Unload the lustre module from node2 (Failover node) by
```
$ lconf -cleanup -node node2 config.xml
```
3. Upgrade the node2 to Lustre 1.4.6 as currently it is NOT running Lustre, and is a backup node.
4. Configure the node2 for Lustre 1.4.6 and failover the node1. Once failed, Lustre will be running from node 2.
5. Unload the lustre module from node1 (Failed node) by
```
$ lconf -cleanup -node node1 config.xml
```
6. Upgrade the node1 to Lustre 1.4.6 as currently it is NOT running the Lustre as we had a failover.
7. Now you can failback to node1 once configured.

CHAPTER II – 9. RAID

9.1 Considerations for Backend Storage

Lustre's architecture allows it to use any kind of block device as backend storage. The characteristics of such devices, particularly in the case of failures vary significantly and have an impact on configuration choices.

This section gives a survey of the issues and recommendations.

9.1.1 Reliability

Given below is a quick calculation that leads to the conclusion that without any further redundancy RAID5 is not acceptable for large clusters and RAID6 is a must.

Take a 1PB file system - that is 2000 disks of 500GB capacity. The MTF of a disk is likely about 1000 days and repair time at 10% of disk bandwidth is close to 1 day (500GB at 5MB/sec = 100,000 sec = 1 day). This means that the expected failure rate is 2000 / 1000 = 2 disks per day.

If we have a RAID5 stripe that is ~10 wide, then during the 1 day of rebuilding the chance that a second disk in the same array fails is about 9 / 1000 ~= 1/100. This means that the in the expected period of 50 days a double failure in a RAID5 stripe will lead to data loss.

So RAID6 or another double parity algorithm is really necessary for OST storage. For the MDS we recommend RAID0+1 storage.

9.1.2 Selecting Storage for the MDS and OSS

The MDS will do a large amount of small writes. For this reason we recommend RAID1 storage. Building RAID1 Linux MD devices and striping over these devices with LVM makes it easy to create an MDS file system of 1-2TB, for example, with 4 or 8 500GB disks.

Having disk monitoring software in place so that rebuilds happen without any delay should be regarded as mandatory. We recommend backups of the meta-data file systems. This can be done with LVM snapshots or using raw partition backups.

We also recommend using a kernel version of 2.6.15 or later with bitmap RAID rebuild features. These reduce RAID recovery time from a rebuild to a quick resynchronization.

9.1.3 Understanding Double Failures with Hardware and Software RAID5

Software RAID does not offer the hard consistency guarantees of top-end enterprise RAID arrays. Those guarantees state that the value of any block is exactly the before or after value and that ordering of writes is preserved. With software RAID, an interrupted write operation that spans multiple blocks can frequently leave a stripe in an inconsistent state that is not restored to either the old or the new value. Such interruptions are normally caused by an abrupt shutdown of the system.

If the array is functioning without disk failures, but experiencing sudden power down events, such interrupted writes on journal file systems can affect file data and data in the journal. Meta data itself is re-written from the journal during recovery and will be correct. Because the journal uses a single block to indicate a complete transaction has committed after other journal writes have completed, the journal remains valid. File data can be corrupted when overwriting file data, but this is a known problem with incomplete writes and caches anyway. Hence recovery of the disk file systems with software RAID is similar to recovery without software RAID. Moreover, using Lustre servers with disk file systems does not change these guarantees.

Problems can arise if after an abrupt shutdown a disk fails on restart. In this case even single block writes provide no guarantee that, for example, the journal will not be corrupted.

Hence:

1. IF A POWERDOWN IS FOLLOWED BY A DISK FAILURE, THE DISK FILE SYSTEM NEEDS A FILE SYSTEM CHECK.

2. IF A RAID ARRAY DOES NOT GUARANTEE before/after SEMANTICS, the same requirement holds.

We believe this requirement is present for most arrays that are used with Lustre, including the successful and popular DDN arrays.

CFS will release a modification to the disk file system that eliminates this requirement for a check with a feature called "journal checksums". With RAID6 this check is not required with a single disk failure, but is required with a double failure upon reboot after an abrupt interruption of the system.

9.1.4 Performance considerations

CFS is currently improving the Linux software RAID code to preserve large I/O which the disk subsystems can do very efficiently. With the existing RAID code software RAID performs equally with all stride sizes, but we expect that fairly large stride sizes will prove advantageous when these fixes are implemented.

9.1.5 Formatting

To format a software RAID file system, use the stride_size option while formatting.

9.2 Disk Performance Measurement

Below are some tips and insights for disk performance measurement. Some of this information is specific to RAID arrays and/or the Linux RAID implementation.

1. Performance is limited by the slowest disk.

Benchmark all disks individually. We have frequently encountered situations where drive performance was not consistent for all devices in the array.

2. Verify drive ordering and identification.

For example, on a test system with a Marvell driver, the disk ordering is not preserved between boots but the controller ordering is. Therefore, we had to perform the sgp_dd survey and create arrays without rebooting.

3. Disks and arrays are very sensitive to request size.

To identify the most ideal request size for a given disk, benchmark the disk with different record sizes ranging from 4 KB to 1-2 MB.

4. By default, the maximum size of a request is quite small.

To properly handle IO request sizes greater than 256 KB, the current Linux kernel either needs a driver patch or some changes in the block layer defaults, namely MAX_SECTORS, MAX_PHYS_SEGMENTS and MAX_HW_SEGMENTS. CFS kernels contain this patch. See blkdev_tunables-2.6-suse.patch in the CFS source.

5. I/O scheduler

Try different I/O schedulers because their behavior varies with storage and load. CFS recommends the deadline or noop schedulers. Benchmark them all and choose the best one for your setup. For further information on I/O schedulers, visit the following URLs:

http://www.linuxjournal.com/article/6931

http://www.redhat.com/magazine/008jun05/features/schedulers/

6. Use the proper block device (sgX versus sdX)

```
size 1048576K rsz 128 crg 8 thr 32 read 20.02 MB/s
size 1048576K rsz 128 crg 8 thr 32 read 56.72 MB/s
```

Both the above outputs were achieved on the same disk with the same parameters for sgp_dd. The only difference is that in the first case /dev/sda was used; while in the second case /dev/sg0 was used. sgX is a special interface that bypasses the block layer and the I/O scheduler, but sends the SCSI commands directly to a drive. sdX is a regular block device, and the requests go through the block layer and the I/O scheduler. The numbers do not change on testing with different I/O schedulers.

7. Requests with partial-stripe write impair RAID5.

Remember that RAID 5 in many cases will do a read-modify-write cycle, which is not performant.

Try to avoid synchronized writes. Probably subsequent writes would make the stripe full and no reads will be needed. Try to configure RAID5 and the application in such a manner that most of the writes will be full-stripe and stripe-aligned.

8. NR_STRIPES in RAID5 (Linux kernel parameter)

This is the size of the internal cache that RAID5 uses for all the operations. If many processes are doing I/O, we suggest you to increase this number. In newer kernels, you can tune it by a module parameter.

9. Do not put an ext3 journal onto RAID5.

As journal is written linearly and synchronously, in most cases writes will not fill whole stripes. In this case, RAID5 will have to read parities.

10. Suggested MD device setups for maximum performance:

MDT

- RAID1 with internal journal and 2 disks from different controllers
- If you require larger MDTs, create 2 equal-sized RAID0 arrays from multiple disks. Create a RAID1 array from these 2 arrays. Using RAID10 directly requires a newer mdadm (the tool that administers software RAID on Linux) than the one shipped with RHEL 4. You can also use LVM instead of RAID0, but this has not been tested.

OST

- File system: RAID5 with 6 disks, each from a different controller.
- External journal: RAID1 with 2 partitions of 400MB (or more), each from disks on different controllers.

```
--mkfsoptions "-j -J device=/dev/mdX"
```

To enable an external journal, you can use the above options in the XML. mdX is the external journal device.

Before running --reformat, run:

```
'mke2fs -O journal_dev -b 4096 /dev/mdX'
```

- You can create a root file system, swap, and other system partitions on a RAID1 array with partitions on any 2 remaining disks. The remaining space on the OST journal disk could be used for this.

CFS has not tested RAID1 of swap.

11. rsz in sgp_dd:

It must be equal to the multiplication of <chunksize> and (disks-1).

You also should pass stripe=N, and extents or mballoc as a mountfs option for OSS. Here N = <chunksize> * (disks-1) / pagesize.

12. Run fsck on power failure or disk failure (RAID arrays).

- You must run fsck on an array in the event of a power failure and failure of a disk in the array due to potential write consistency issues.
- You can automate this in rc.sysinit by detecting degraded arrays.

9.2.1 Sample Graphs

9.2.1.1 Graphs for Write Performance:

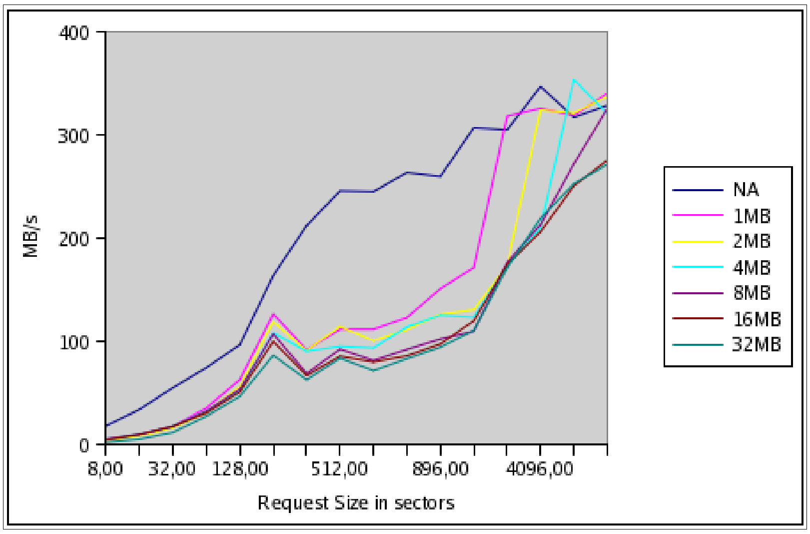

Figure 2.9.1: Write - RAID0, 64K chunks, 6 spindles

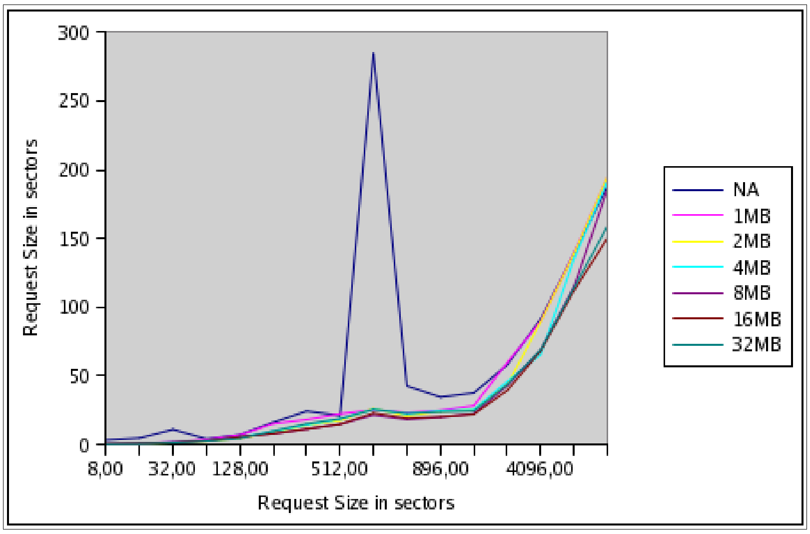

Figure 2.9.2: Write - RAID5, 64K chunks, 6 spindles

9.2.1.2 Graphs for Read Performance:

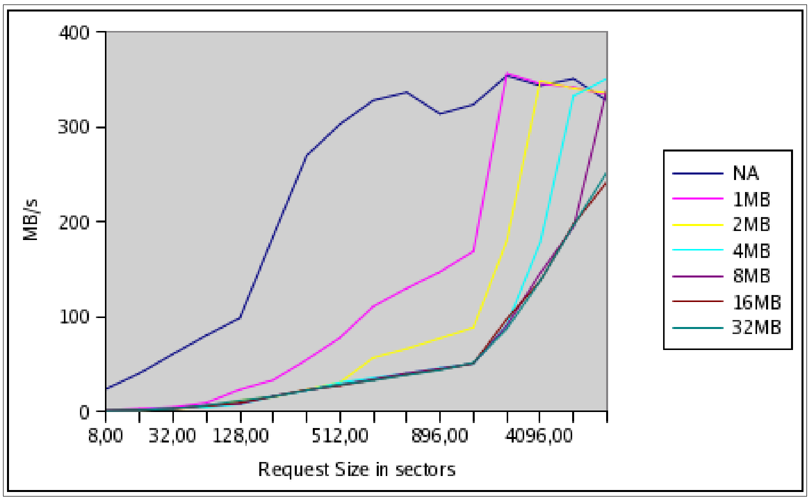

Figure 2.9.3: Read - RAID0, 64K chunks, 6 spindles

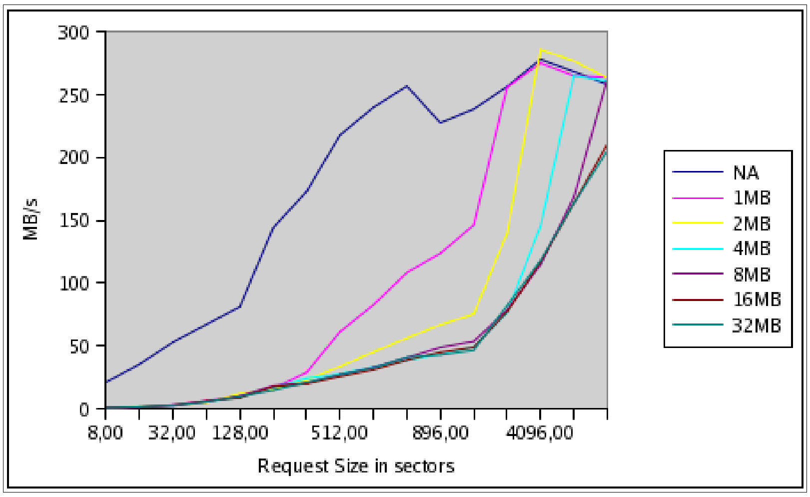

Figure 2.9.4: Read – RAID5, 64 K chunks, 6 spindle

Chapter II – 10. Bonding

10.1 Network Bonding

Bonding is a method of aggregating multiple physical links into a single logical link. This technology is also known as trunking, port trunking and link aggregation. We will use the term bonding.

Several different types of bonding are supported in Linux. All these types are referred to as 'modes', and use the 'bonding' kernel module.

Modes 0 to 3 provide support for load balancing and fault tolerance by using multiple interfaces. Mode 4 aggregates a group of interfaces into a single virtual interface where all members of the group share the same speed and duplex settings. This mode is described under IEEE spec 802.3ad, and it is referred to as either 'mode 4' or '802.3ad'.

(802.3ad refers to mode 4 only. The detail is contained in Clause 43 of the IEEE 8 - the larger 802.3 specification. Consult IEEE for more information.)

10.2 Requirements

The most basic requirement for successful bonding is that both endpoints of the connection must support bonding. In a normal case, the non-server endpoint is a switch. (Two systems connected via crossover cables can also use bonding.) Any switch used must explicitly support 802.3ad Dynamic Link Aggregation.

The kernel must also support bonding. All supported Lustre kernels have this support. The network driver for the interfaces to be bonded must have the ethtool support. The ethtool support is necessary for determination of the slave speed and duplex settings. All recent network drivers implement it.

To verify that your interface supports ethtool:

```
$ which ethtool

$ ethtool eth0

Settings for eth0:

Supported ports: [ MII ]

Supported link modes:   10baseT/Half 10baseT/Full \
100baseT/Half 100baseT/Full1000baseT/ \
Half1000baseT/Full

Supports auto-negotiation: Yes

(ethtool will return an error if your card is not
supported.)
```

To quickly check whether your kernel supports bonding:

```
$ grep ifenslave /sbin/ifup

$ which ifenslave
```

NOTE: Bonding and ethtool have been available since 2000. All Lustre-supported kernels include this functionality.

10.3 Bonding Module Parameters

Bonding Module Parameters control various aspects of bonding.

Outgoing traffic is mapped across the slave interfaces according to the transmit hash policy. For Lustre, we recommend setting the xmit_hash_policy option to the 'layer3+4' option for bonding. This policy uses upper layer protocol information if available to generate the hash. This allows traffic to a particular network peer to span multiple slaves, although a single connection does not span multiple slaves. :

```
$ xmit_hash_policy=layer3+4
```

The 'miimon' option enables users to monitor the link status. (The parameter is a time interval in milliseconds.) It makes the failure of an interface transparent to avoid serious network degradation during link failures. 100 milliseconds is a reasonable default. Increase the timeout for a busy network.

```
$ miimon=100
```

10.4 Setup

Follow the process below to setup bonding:

Create a virtual 'bond' interface.

Assign an IP address to the 'bond' interface.

Attach one or more 'slave' interfaces to the 'bond' interface. Typically the MAC address of the first slave interface will become the MAC address of the bond.

Setup the bond interface and its options in /etc/modprobe.conf. Start the slave interfaces by your normal network method.

> NOTE: You must modprobe the bonding module for each bonded interface. If you wish to create bond0 and bond1, two entries in modprobe.conf are required.

Our examples are from Red Hat systems, and use /etc/sysconfig/networking-scripts/ifcfg-* for setup. The OSDL reference site given below includes detailed instructions for other configuration methods, instructions for using DHCP with bonding, and other setup details. We strongly recommend using this site.

http://linux-net.osdl.org/index.php/Bonding

Check /proc/net/bonding to determine status on bonding. There should be a file there for each bond interface. Check the interface state with ethtool or ifconfig. ifconfig lists the first bonded interface as 'bond0'.

10.4.1 Examples

Let us see an example of Modprobe.conf for bonding ethernet interfaces eth1 and eth2 to bond0:

```
install bond0 /sbin/modprobe -a eth1 eth2 &&  \
/sbin/modprobe bonding \
miimon=100 mode=802.3ad xmit_hash_policy=layer3+4

alias bond0 bonding
```

ifcfg-bond0

```
DEVICE=bond0

BOOTPROTO=static

IPADDR=###.###.##.##
(Assign here the IP of the bonded interface.)

NETMASK=255.255.255.0

ONBOOT=yes
```

ifcfg-eth1 (eth2 is a duplicate)

```
DEVICE=eth1 # Change to match device
```

```
MASTER=bond0

SLAVE=yes

BOOTPROTO=none

ONBOOT=yes

TYPE=Ethernet
```

From linux-net.osdl.org:

```
For example, the content of /proc/net/bonding/bond0
after the driver is loaded with parameters of mode=0
and miimon=1000 is generally as follows:

   Ethernet Channel Bonding Driver: 2.6.1 (October
29, \ 2004)

        Bonding Mode: load balancing (round-robin)

        Currently Active Slave: eth0

        MII Status: up

        MII Polling Interval (ms): 1000

        Up Delay (ms): 0

        Down Delay (ms): 0


        Slave Interface: eth1

        MII Status: up

        Link Failure Count: 1


        Slave Interface: eth0

        MII Status: up

        Link Failure Count: 1
```

In the example below, the bond0 interface is the master (MASTER) while eth0 and eth1 are slaves (SLAVE).

> NOTE: All the slaves of bond0 have the same MAC address (Hwaddr) – bond0. All modes except TLB and ALB have this MAC address. TLB and ALB require a unique MAC address for each slave.

```
 $ /sbin/ifconfig

bond0     Link encap:Ethernet  Hwaddr \
00:C0:F0:1F:37:B4

         inet addr:XXX.XXX.XXX.YYY \
Bcast:XXX.XXX.XXX.255  Mask:255.255.252.0

         UP BROADCAST RUNNING MASTER MULTICAST \
MTU:1500  Metric:1

         RX packets:7224794 errors:0 dropped:0 \
overruns:0 frame:0
```

```
          TX packets:3286647 errors:1 dropped:0 \
overruns:1 carrier:0
          collisions:0 txqueuelen:0

 eth0      Link encap:Ethernet  Hwaddr \
00:C0:F0:1F:37:B4
          inet addr:XXX.XXX.XXX.YYY \
Bcast:XXX.XXX.XXX.255  Mask:255.255.252.0
          UP BROADCAST RUNNING SLAVE MULTICAST \
MTU:1500  Metric:1
          RX packets:3573025 errors:0 dropped:0 \
overruns:0 frame:0
          TX packets:1643167 errors:1 dropped:0 \
overruns:1 carrier:0
          collisions:0 txqueuelen:100
          Interrupt:10 Base address:0x1080

 eth1      Link encap:Ethernet  Hwaddr \
00:C0:F0:1F:37:B4
          inet addr:XXX.XXX.XXX.YYY \
Bcast:XXX.XXX.XXX.255  Mask:255.255.252.0
          UP BROADCAST RUNNING SLAVE MULTICAST \
MTU:1500  Metric:1
          RX packets:3651769 errors:0 dropped:0 \
overruns:0 frame:0
          TX packets:1643480 errors:0 dropped:0 \
overruns:0 carrier:0
          collisions:0 txqueuelen:100
          Interrupt:9 Base address:0x1400
```

10.5 Lustre Configuration

Lustre uses the IP address of the bonded interfaces and requires no special configuration. It treats the bonded interface as a regular TCP/IP interface. If necessary, specify 'bond0' using the Lustre 'networks' parameter:

```
options lnet networks=tcp(bond0)
```

10.6 References

Below are some references that we recommend -

- In the Linux kernel source tree, see

  Documentation/networking/bonding.txt
- http://linux-ip.net/html/ether-bonding.html
- http://www.sourceforge.net/projects/bonding

  This is the bonding sourceforge site.
- http://linux-net.osdl.org/index.php/Bonding

  This is the most exhaustive reference and is highly recommended. It includes explanations of more complicated setups, including the use of DHCP with bonding.

Part III. Lustre Tuning, Monitoring and Troubleshooting

CHAPTER III – 1. LUSTRE I/O KIT

1.1 Prerequisites

Lustre IO-Kit is a collection of benchmark tools for a cluster with the Lustre file system. Currently only an Object Block Device-survey is included, but the kit may be extended with blockdevice- and file system- survey in future. The IO-Kit can be downloaded from

https://downloads.clusterfs.com/customer/lustre-iokit/

The Prerequisites for the I/O kit are:

- python2.2 or newer, available at /usr/bin/python2
- The "logging"-module from python2.3
- Passwordless remote access to nodes in the system, through
- ScaSH (The Scali parallel shell) or SSH
- Lustre File System
- sg3_utils for the sgp_dd utility

The kit can be used to validate the performance of the various hardware and software layers in the cluster, and also as a way of tracking down and troubleshooting I/O issues. It is very important to establish performance from the “bottom up” perspective. Performance of a single raw device should be verified first and then it should be verified that the performance is stable with a larger number of devices. Frequently while troubleshooting such performance issues, we find that array performance with all LUNs running does not match the performance of a single LUN tested in isolation. After the raw performance has been established, the other software layers can be added and tested in an incremental manner.

There are three tests in the kit, the first of the tests surveys basic performance of the device and bypasses the kernel block device layers, buffer cache and file system. The subsequent surveys test progressively higher layers of the Lustre stack. Typically, with these tests, Lustre should deliver 85-90% of the raw device performance.

1.2 Interfaces

Two interfaces to OBD survey; a python library interface and a command line interface. The python library interface is documented in pydoc. Use "pydoc lustre_obdsurveylib" for detailed information about the interface.

The command line interface is accessed through the "lustre_obdsurvey" application. Running the application with "--help" lists the various options.

1.3 Running the I/O Kit Surveys

The I/O Kit bundle contains two tools: -

- sgpdd survey
- obd survey

1.3.1 sgpdd-survey

This is the tool for testing of the 'bare metal' performance, bypassing as much of the kernel as we can. It does not require Lustre, but does require the sgp_dd package. This survey may be used to characterize the performance of a SCSI device. It simulates an OST serving multiple stripe files. The data gathered by it can help set expectations for the performance of a lustre OST exporting the device.

The script uses sgp_dd to do raw sequential disk I/O. It runs with variable numbers of sgp_dd threads to show how performance varies with different request queue depths.

The script spawns variable numbers of sgp_dd instances, each reading or writing a separate area of the disk to show how performance varies with the number of concurrent stripe files.

The script must be customized according to the particular device under test and where it should keep its working files. Customization variables are described explicitly at the start of the script.

When the script runs, it creates a number of working files and a pair of result files. All files start with the prefix given by ${rslt}.

```
${rslt}_<date/time>.summary same as stdout

${rslt}_<date/time>_* tmp files

${rslt}_<date/time>.detail collected tmp files for
post-mortem
```

The summary file and stdout contain lines like...

total_size 8388608K rsz 1024 thr 1 crg 1 180.45 MB/s 1 x 180.50 =/ 180.50 MB/s

The number immediately before the first MB/s is the bandwidth computed by measuring total data and elapsed time. The other numbers are a check on the bandwidths reported by the individual sgp_dd instances.

If there are so many threads that sgp_dd is unlikely to be able to allocate I/O buffers, "ENOMEM" is printed.

If all the sgp_dd instances do not successfully report a bandwidth number, "failed" is printed.

CHAPTER III – 2. LUSTREPROC

2.1 Introduction

The proc file system acts as an interface to internal data structures in the kernel. It can be used to obtain information about the system and to change certain kernel parameters at runtime (sysctl).

The Lustre file system provides several proc file system variables that control aspects of Lustre performance and provide information.

The proc variables are classified based on the subsystem they affect.

2.1.1 /proc entries for Lustre

2.1.1.1 Recovery

4 **/proc/sys/lustre/upcall**

This will contain the path of the recovery upcall,or DEFAULT for the normal case where there is no upcall. Certain states will place information here, including

- FAILED_IMPORT – tgt_uuid obd_uuid net_uuid – indicates failure of an upcall the uuid information identifies target, obd name and network.
- RECOVERY_OVER tgt_uuid – This is the upcall called on the server when the recovery period has ended. The uuid is the target that was in recovery mode. EXAMPLE syslog message:

```
"May 25 13:35:46 d2_q_0 kernel: Lustre:
12162:0:(recover.c:77:ptlrpc_run_recovery_over_upcall
()) Invoked upcall DEFAULT RECOVERY_OVER ost-
alpha_UUID"
```

5 /**proc/sys/lnet/upcall**

- LBUG src_file line_number function – This is called when an LBUG occurs.

The script paths can be configured with lmc and/or lconf or by modifying the corresponding *proc* entries. Setting an upcall to "DEFAULT" means that the recovery will be handled within the kernel by reconnecting to the same device.

2.1.1.2 Lustre Timeouts/ Debugging

1. **/proc/sys/lustre/timeout** – This is the time period for which a client will wait on a server to complete an RPC (Default 100s). Servers will wait 1/2 of this time for a normal client RPC to complete, and 1/4th of this time for a single bulk

request (read or write of up to 1MB) to complete. The client will ping recoverable targets (MDS and OSTs) at 1/4th of the timeout, and the server will wait 1.5 times the timeout before evicting a client for being "stale".

2. **/proc/sys/lustre/ldlm_timeout** – This is the time period for which a server will wait for a client to reply to an initial AST (lock cancellation request) (Default is 20s for an OST and 6s for an MDS). If the client replies to the AST the server will give it a *norma*l timeout (1/2 of the client timeout) to flush any dirty data and release the lock.

3. **/proc/sys/lustre/fail_loc** - Internal debugging failure hook. See lustre/include/linux/obd_support.h for the definitions of individual failure locations. Default value is 0.

```
sysctl -w lustre.fail_loc=0x80000122 # drop a single
reply
```

4. **/proc/sys/lustre/dump_on_timeout** – Triggers dumps of the lustre debug log when timeouts occur

2.2 I/O

1. **/proc/fs/lustre/llite/fs0/max_read_ahead_mb** – Size of the client per-file read-ahead (default 40 MB). Setting this to 0 will disable readahead.

2. **/proc/fs/lustre/llite/fs0/max_cache_mb** – Maximum amount of inactive data cached by the client (default 3/4 of RAM).

2.2.1 Client I/O RPC Stream Tunables

The Lustre engine will always attempt to pack an optimal amount of data into each I/O RPC and will attempt to keep a consistent number of issued RPCs in progress at a time. Lustre exposes several tuning variables to adjust behaviour according to network conditions and cluster size. Each OSC has its own tree of these tunables. For example:

```
$ ls -d
/proc/fs/lustre/osc/OSC_client_ost1_MNT_client_2\
/localhost

/proc/fs/lustre/osc/OSC_uml0_ost1_MNT_localhost

/proc/fs/lustre/osc/OSC_uml0_ost2_MNT_localhost

/proc/fs/lustre/osc/OSC_uml0_ost3_MNT_localhost

$ ls /proc/fs/lustre/osc/OSC_uml0_ost1_MNT_localhost

blocksize           filesfree           max_dirty_mb
ost_server_uuid             stats
```

... and so on

The files related to tuning the RPC stream are as follows:

max_dirty_mb

This controls how many megabytes of dirty data can be written and queued up in the OSC. POSIX file writes that are cached contribute to this count. When the limit is reached additional writes will stall until previously cached writes are written to the server. This may be changed by writing a single ASCII integer to the file. Only values between 0 and 512 are allowed. If 0 is given, no writes will be cached, but unless you use large writes (1MB or more) performance will suffer noticably.

cur_dirty_bytes

This is a read-only value that returns the current amount of bytes written and cached on this OSC.

max_pages_per_rpc

This value represents the maximum number of pages that will undergo I/O in a single RPC to the OST. The minimum is a single page and the maximum for this setting is platform depedent (256 for i386/x86_64, possibly less for ia64/PPC with larger PAGE_SIZE), though generally amounts to a total of 1 megabyte in the RPC.

max_rpcs_in_flight

This value represents the maximum number of concurrent RPCs that the OSC will issue at a time to its OST. If the OSC tries to initiate an RPC but finds that it already has these many RPCs outstanding, it will wait to issue further RPCs until some complete. The minimum is 1 and maximum is 32 for this setting.

The value for max_dirty_mb is recommended to be 4 * max_pages_per_rpc * max_rpcs_in_flight in order to maximize performance.

> NOTE: On an MDS, the path to the files above is - /proc/fs/lustre/osc/OSC_mds_ost1_mds1. Whereas the path on a Client is - /proc/fs/lustre/osc/OSC_client_ost1_MNT_client_2.

2.2.2 Watching the Client RPC Stream

In the same directory is a file that gives a histogram of the make-up of previous RPCs.

```
# cat /proc/fs/lustre/osc \
/OSC_uml0_ost1_MNT_localhost/rpc_stats
snapshot_time:         1067551484:37103 (secs:usecs)
RPCs in flight:        0
pending write pages:   0
pending read pages:    0

other RPCs in flight when a new RPC is sent:
0:              0
1:              0
2:              0
3:              0
4:              0
5:              0
6:              0
7:              0
8:              0
9:              0
```

```
10:            0
11:            0
12:            0
13:            0
14:            0
15:            0

pages in each RPC:
0:             0
1:             0
2:             0
3:             0
4:             0
5:             0
6:             0
7:             0
8:             0
9:             0
10:            0
11:            0
12:            0
13:            0
14:            0
15:            0
```

RPCs in flight

Not surprisingly, this represents the number of RPCs that are issued by the OSC but are not complete at the time of the snapshot. It should always be less than or equal to **max_rpcs_in_flight**.

pending {read,write} pages

These show the number of pages that have been queued for I/O in the OSC.

other RPCs in flight when a new RPC is sent

As an RPC is sent, it records the number of other RPCs that were pending in this table. If in case the RPC being sent is the first RPC, the 0: row will be incremented. If this new RPC is sent while another is pending the 1: row will be incremented, and so on. The number of RPCs that are pending as each RPC *completes* is not tabulated. This table is a good way of visualizing the concurrency of the RPC stream. One hopes to see a large clump around the **max_rpcs_in_flight** value which shows that the network is being kept busy.

pages in each RPC

As an RPC is sent, the order of the number of pages it is made of is recorded in this table. A single page RPC increments the 0: row, 128 pages the 7: row, and so on.

These histograms can be cleared by writing any value into the *rpc_stats* file.

2.2.3 Watching the OST Block I/O Stream

Similarly, there is a "brw_stats" histogram in the obdfilter directory which shows you the statistics for number of I/O requests sent to the disk, their size, and whether they are contiguous on disk or not.

```
cat /proc/fs/lustre/obdfilter/OST_localhost/brw_stats
snapshot_time:         1089922302:248138 (secs:usecs)

                    read                    write
pages per brw     brws   % cum % |    rpcs    %  cum   %
1:                   0   0   0   |       1    0   0
2:                   0   0   0   |       0    0   0
4:                   0   0   0   |       0    0   0
8:                   0   0   0   |       0    0   0
16:                  0   0   0   |       0    0   0
32:                  0   0   0   |       0    0   0
64:                  0   0   0   |       0    0   0
128:                 0   0   0   |     140   99 100

                    read                    write
discont pages     rpcs   % cum % |    rpcs    %  cum   %
0:                   0   0   0   |     141  100 100

                    read                    write
discont blocks    rpcs   % cum % |    rpcs    %  cum   %
0:                   0   0   0   |     123   87  87
1:                   0   0   0   |      18   12 100
```

pages per brw – Number of pages per RPC request (should match aggregate client rpc_stats)

discont pages – Number of discontinuities in the logical file offset of each page in a single RPC

discont blocks – Number of discontinuities in the physical block allocation in the filesystem for a single RPC

2.2.4 mballoc History

/proc/fs/ldiskfs/loop0/mb_history

Example:

```
 goal           result         found  grps  cr  merge
tail  broken

138/18088/97  138/14986/97    31      1    1
43    64

138/20297/36  138/16101/36    31      1    1
9     64

138/18019/166 138/15456/166   31      1    1
6     16

138/19309/115 138/14848/115   31      1    1
115   512
```

Fields:

goal – Request that came to mballoc (group/block-in-group/number-of-blocks)

result – What mballoc actually found for the request

found – Number of free chunks mballoc found and measured before the final decision

grps – Number of groups mballoc scanned to satisfy the request

cr – Stage, at which mballoc found the result:

- **0** – The best in terms of resource allocation took (CPU), used for large requests like 1 or more MBs
- **1** – Regular stage (good at resource consumption)
- **2** – fs is quite fragmented (not that bad at resource consumption)
- **3** – fs is very fragmented (worst at resource consumption)

merge – Whether the request hit the goal. This is good, because then extents code can merge new blocks to existing extent, so extents tree does not need to grow

tail – Number of blocks left free after the allocation breaks large free chunk

broken – How large the broken chunk was

Most customers are probably interested in **found/cr**. If **cr** is 0 or 1 and **found** is less than 100, then mballoc is doing quite well.

Also, number-of-blocks-in-request (3rd number in the goal triple) can tell the number of blocks requested by the obdfilter. If the obdfilter is doing a lot of small requests (just few blocks), then either the client is doing I/O to a lot of small files, or something may be wrong with the client because it is better if client sends large I/O requests. This can be investigated with the OSC rpc_stats, or OST brw_stats mentioned above.

Number of groups scanned (grps column) should be small. if it reaches few dozens often – either your disk filesystem is pretty fragmented or mballoc is doing wrong in the group selection part.

2.3 Locking

/proc/fs/lustre/ldlm/ldlm/namespaces/<OSC name|MDC name>/lru_size – This variable determines how many locks can be queued up on the client in an LRU queue. The default value of LRU size is 100. Increasing this on a large number of client nodes is not recommended, though servers have been tested with up to 150,000 total locks (num_clients * lru_size). Increasing it for a small number of clients (e.g. login nodes with a large working set of files due to interactive use) can speed up lustre dramatically. Recommended values are in the neighbourhood of 2500 MDC locks and 1000 locks per OSC.

The following command can be used to clear the LRU on a single client (and as a result flush client cache) without changing the LRU size value:

```
$ echo clear > \
/proc/fs/lustre/ldlm/ldlm/namespaces/<OSC name|MDC \
name>/lru_size
```

If you shrink the LRU size below the number of existing unused locks, they are cancelled immediately. Echo "clear" to cancel all locks without changing the value.

2.4 Debug Support

1. **/proc/sys/lnet/debug** – Setting this to 0 will completely turn-off debug logs for all the debug types. While setting it to -1 will turn on full debugging (see D_* definitions in lnet/include/linux/libcfs.h).

2. **/proc/sys/lnet/subsystem_debug** – This controls the debug logs for subsystems (see S_* definitions).

3. **/proc/sys/lnet/debug_path** – This indicates the location where debugging symbols should be stored for gdb. The default is set to */r/tmp/lustre-log-localhost.localdomain*.

These values can also be set via **sysctl -w lnet.debug={value}**.

NOTE: Above entries exist only when Lustre has already been loaded.

Lustre uses the set debug level after it is loaded on a particular node. You can set the debug level by adding the following to the node entry config shell script:

```
--ptldebug <level>
```

2.4.1 RPC Information For Other OBD Devices

Some OBD devices maintain a count of the number of RPC events that they process. Sometimes these events are more specific to operations of the device, like *llite*, than actual raw RPC counts.

```
$ find /proc/fs/lustre/ -name stats
/proc/fs/lustre/llite/fs0/stats
/proc/fs/lustre/mdt/MDT/mds_readpage/stats
/proc/fs/lustre/mdt/MDT/mds_setattr/stats
/proc/fs/lustre/mdt/MDT/mds/stats
/proc/fs/lustre/osc/OSC_uml0_ost3_MNT_localhost/stats
/proc/fs/lustre/osc/OSC_uml0_ost2_MNT_localhost/stats
/proc/fs/lustre/osc/OSC_uml0_ost1_MNT_localhost/stats
/proc/fs/lustre/osc/OSC_uml0_ost3_mds1/stats
/proc/fs/lustre/osc/OSC_uml0_ost2_mds1/stats
/proc/fs/lustre/osc/OSC_uml0_ost1_mds1/stats
/proc/fs/lustre/obdfilter/ost2/stats
/proc/fs/lustre/obdfilter/ost3/stats
/proc/fs/lustre/obdfilter/ost1/stats
/proc/fs/lustre/ost/OSS/ost_create/stats
```

```
/proc/fs/lustre/ost/OSS/ost/stats
/proc/fs/lustre/ldlm/ldlm/ldlm_canceld/stats
/proc/fs/lustre/ldlm/ldlm/ldlm_cbd/stats
```

The OST .../stats files can be used to track the performance of RPCs that the OST gets from all clients, for example. It is possible to get a periodic dump of values from these files (For instance, every 10s) that show the RPC rates (similar to iostat) by using the "llstat.pl" tool like:

```
$ llstat.pl /proc/fs/lustre/ost/OSS/ost/stats 10

/proc/fs/lustre/ost/OSS/ost/stats @ 1126198063.790389

Name                 Cur.Count  Cur.Rate   #Events
Unit       last     min          avg      max    stddev

req_waittime         12         0        1522
[usec]  19800.50    68       1135.52  242393   10297.09

req_qdepth           12         0        1522
[reqs]      0.58     0          0.15       3       0.45

req_active           12         0        1522
[reqs]      1.08     1          1.01       2       0.09

reqbuf_avail         12         0        1522
[bufs]     63.67    63         63.93      64       0.26

ost_setattr           0         0           2
[usec]      0.00    240        257.50     275      24.75

ost_read              0           0       220
[usec]      0.00    530      1262.77   74463    4972.71

ost_write             0           0           230
[usec]      0.00   1438      2200.02   28189    2342.42

ost_create            2           0           24
[usec]    274.00    72       7322.46   35521   12654.60

ost_destroy         400          18          1047
[usec]    736.09   626       1134.41   30260    1560.68

ost_get_info          0           0           2
[usec]      0.00    71        101.50     132      43.13

ost_connect           2           0        26
[usec]   1395.50  1170        5037.04   27153    7231.62

ost_set_info          2           0        24
[usec]    297.50   108        300.38    1162     208.49

ldlm_enqueue          0           0       474
[usec]      0.00   194        351.57    1911     154.21

obd_ping              4           0           294
[usec]    151.50    62        175.97      600      49.36
```

Cur.Count – It is the number of events of each type sent in the last interval (10s in this case)

Cur.Rate – It is the number of events per second in the last interval

#Events – It is the total number of such events since the system was started

Unit – It is the unit of measurement for that statistic (microseconds, requests, buffers).

last – This shows the average rate of these events (in units/event) for the last interval during which they arrived (For instance, in the above mentioned case of ost_destroy it took an average of 736 microseconds per destroy for the 400 object destroys in the previous 10s.

min – This is the minimum rate (in units/event) since the service started.

avg – This is the average rate.

max – It is the maximum rate.

stddev – It reports the standard deviation (not measured in all the cases).

The events common to all services are:

req_waittime is the amount of time a request waited in the queue before being handled by an available server thread

req_qdepth is the number of requests waiting to be handled in the queue for this service

req_active is the number of requests currently being handled

reqbuf_avail is the number of unsolicited lnet request buffers for this service

Some service specific events of interest are:

ldlm_enqueue is the time it takes to enqueue a lock (this includes file open on the MDS)

mds_reint is the time it takes to process an MDS modification record (includes create, mkdir, unlink, rename, setattr)

Chapter III – 3. Lustre Tuning

3.1 Module Options

Following option allows the number of OST service threads to be specified at module load time on the OSS nodes:

```
$ vi /etc/modprobe.conf:
```

Find and modify "ost ost_num_threads" in the file above with the following change -

options ost ost_num_threads={N}

The default number of OSS service threads on the OSS depends on the server size up to a maximum of 36. Tests have shown that increasing the number of threads can improve I/O performance when many clients are doing concurrent I/O. The maximum number of threads (compile time constant OST_MAX_THREADS) is 512.

Consider several OSTs are exported from a single OSS, or the back-end storage is running synchronously and/or I/O completion is taking more time. In such cases, a larger number of I/O threads allows the kernel and storage to aggregate many writes together for more efficient disk I/O. The OST thread pool is shared. Each thread allocates approximately 1.5MB (maximum RPC size + 0.5MB) for internal I/O buffers. So consider this along with the total OSS RAM size.

In **Lustre 1.4.7** there is a similar parameter for the number of MDS service threads:

$ vi /etc/modprobe.conf:

options mds mds_num_threads={N}

No testing has been done at this time what the optimal number of MDS threads are. The default number varies based on the server size up to a maximum of 32. The maximum number of threads (MDS_MAX_THREADS) is 512.

3.2 DDN Tuning

This section gives a guideline to configure DDN storage arrays for use with Lustre.

3.2.1 Settings

3.2.1.1 Segment Size

The cache segment size affects the I/O performance noticeably. Set it differently on the MDT (which does small, random I/O) and an OST (which does large, contiguous I/O). The optimum values found in customer testing are 64KB for the MDT and 1MB for the OST:

The necessary DDN client commands are given below.

For MDT LUN -

```
$ cache size=64

size is in KB, 64, 128, 256, 512, 1024, and 2048.
Default 128
```

For OST LUN -

```
$ cache size=1024
```

3.2.1.2 maxcmds

In a particular case, changing this value from the default 2 to 4 has improved the write performance by as good as 30%. This works only with SATA-based disks and when only one controller of the pair is actually accessing the shared LUNs

This information comes with a warning, as DDN support do not recommend changing this setting from the default. By increasing the value to 5, the same set up experienced some serious problems.

The necessary DDN client command is given below.

(The default value is 2.)

```
$ disk   maxcmds=3
```

3.2.1.3 Write-back cache

Some customers run with the write-back cache turned on, because it improves performance noticeably. They are willing to take the risk that, when there is a DDN controller crash and they need to run e2fsck, it will take them less time than the performance hit from running with the write-back cache turned off.

Other customers run with the write-back cache off, for increased data security. However, some of these customers experience performance problems with the small writes during journal flush. In this mode it is highly beneficial to also increase the number of OST service threads "ost_num_threads=512" in /etc/modprobe.conf, if the OST has enough RAM (about 1.5MB/thread is preallocated for IO buffers). More I/O threads allow more I/O requests to be in flight and waiting for the disk to complete the synchronous write.

This is a decision that you need to make yourself - Is performance more important than the slight risk of data loss along with a downtime in case of a hardware/software problem on the DDN. Note that there is no risk from an OSS/MDS node crashing, only if the DDN itself fails.

3.2.1.4 Further Tuning Tips

Experiences drawn from testing at a large installation:

- Separate the EXT3 OST into 2 LUNs, a small LUN for the EXT3 journal, and a big one for the "data"
- Since Lustre 1.0.4, we can supply EXT3 mkfs options when we create the OST like -j -J and so on in the following manner – (where /dev/sdj has been formatted before as a journal)

```
$ {LMC} --add mds --node io1 --mds iap-mds --dev
/dev/sdi --mkfsoptions "-j -J device=/dev/sdj" --
failover --group iap-mds
```

Very important: We have proved that we need to create one OST per TIER especially in Write Through (see the illustration below). This matters if you have 16 Tiers: Create 16 OST one Tier each instead of 8 made of 2 Tiers each.

You are not obliged to lock in cache the small LUNs.

Illustration - One OST per Tier

LUN	Label	Owner	Status	Capacity (Mbytes)	Block Size	Tiers	Tier list
0		1	Ready	102400	512	1	1
1		1	Ready	102400	512	1	2
2		1	Ready	102400	512	1	3
3		1	Ready	102400	512	1	4
4		2	Ready [GHS]	102400	4096	1	5
5		2	Ready [GHS]	102400	4096	1	6
6		2	Critical	102400	512	1	7
7		2	Critical	102400	4096	1	8
10		1	Cache Locked	64	512	1	1
11		1	Cache Locked	64	512	1	2
12		1	Cache Locked	64	512	1	3
13		1	Cache Locked	64	512	1	4

14	2	Ready [GHS]	64	512	1	5
15	2	Ready [GHS]	64	512	1	6
16	2	Critical	64	512	1	7
17	2	Critical	64	512	1	8

System verify extent: 16MB

System verify delay: 30

Part IV. Lustre for Users

Chapter IV – 1. Free Space and Quotas

1.1 Querying File System Space

To determine the disk space available on a file system, df is used. It displays the amount of available disk space on the file system for each given file name argument. If no file name is given, the space available on all the currently mounted file systems is shown. The disk space is shown in 1K blocks by default. But if the environment variable POSIXLY_CORRECT is set, 512-byte blocks are used. If an argument is the absolute file name of a disk device node containing a mounted file system, df shows the space available on that file system rather than on the file system containing the device node (which is always the root file system). This version of df cannot show the space available on unmounted file systems. This is because in case of most types of systems, doing so requires very non-portable intimate knowledge of file system structures.

Options	Description
-a, --all	Includes file systems having 0 blocks
-B	block-size=SIZE uses SIZE-byte blocks
-h	--human-readable print sizes in human readable format (e.g., 1K 234M 2G)
-H, --si	--human -readable. Similar, but uses powers of 1000 and not 1024.
-i, --inodes	Lists inodes information instead of block usage

1.2 Using Quota

The quota command displays the disk usage and quotas. By default, or with the **-u** flag, only user quotas are displayed. The quota command reports the quotas of all the file systems listed in the file /etc/filesystems. If the quota command exits with a non-zero status, one or more file systems are over quota.

A root user may use the **-u** flag with the optional *User* parameter to view the limits of other users. Users without the root user authority can view the limits of groups (of which they are members) by using the **-g** flag with the optional *Group* parameter.

> Note: If a particular user has no files in a file system on which that user has a quota, this command displays "*quota: none*" for that user. The user's actual quota is displayed when the user has files in the file system.

Examples

To display your quotas as a user "bob", enter:

```
$ lfs quota -u  /mnt/lustre
```

The example above will display the disk usage and limits for the user "bob".

To display quotas as the root user for user "bob", enter:

```
$lfs quota -u bob /mnt/lustre
```

The system displays following information about the disk user by "bob":

To display your group's quota as "tom":

```
$ lfs -g tom /mnt/lustre
```

To display the group's quota of "tom":

```
$lfs quota -g tom /mnt/lustre
```

Chapter IV – 2. Striping and Other I/O Options

2.1 File Striping

Lustre stores files in one or more objects on object storage targets (OSTs). When a file is comprised of more than one object, Lustre will stripe the file data across them in a round-robin fashion. The number of stripes, the size of each stripe, and the servers chosen are all configurable.

One of the most frequently-asked Lustre questions is *“How should I stripe my files, and what is a good default?”* The short answer is that it depends on your needs. A good rule of thumb is to stripe over as few objects as will meet those needs, and no more.

2.1.1 Advantages of Striping

There are two reasons to create files of multiple stripes: bandwidth and size.

There are many applications which require high-bandwidth access to a single file – more bandwidth than can be provided by a single OSS – for example, scientific applications which write to a single file from hundreds of nodes, or a binary executable which is loaded by many nodes when an application starts.

In cases such as these, you want to stripe your file over as many OSSs as it takes to achieve the required peak aggregate bandwidth for that file. In our experience, the requirement is “as quickly as possible”, which usually means all OSSs.

> NOTE: This assumes that your application is using enough client nodes, and can read/write data fast enough, to take advantage of that much OSS bandwidth. The largest useful stripe count is bounded by (the I/O rate of your clients/jobs) divided by (the performance per OSS).

The second reason to stripe is when a single object storage target (OST) does not have enough free space to hold the entire file.

2.1.2 Disadvantages of Striping

There are two disadvantages to striping, which should deter you from choosing a default policy which stripes over all OSTs unless you really need it: increased overhead and increased risk.

Increased overhead comes in the form of extra network operations during common operations such as stat and unlink, and more locks. Even when these operations can be performed in parallel, there is a big difference between doing one network operation and doing one hundred.

Increased overhead also comes in the form of server concurrency. Consider a cluster with 100 clients and 100 OSSs, each with 1 OST. If each file has exactly one object and the load is distributed evenly, there is no concurrency, and the disks on each server can manage sequential I/O. If each file has 100 objects, then the clients will all compete with each other for the attention of the servers, and the disks on each node will be seeking in 100 different directions. In this case, there is needless concurrency.

Increased risk is evident when you consider again the example of striping each file across all servers. In this case, if any one OSS catches on fire, a small part of every file will be lost. By comparison, if every file has exactly one stripe, you will lose fewer files, but you will lose them in their entirety. Most users would rather lose some of their files entirely than all of their files partially.

2.1.3 Stripe Size

Choosing a stripe size is a small balancing act, but there are reasonable defaults. The stripe size must be a multiple of the page size; for safety, Lustre tools enforce a multiple of 16 KB (the page size on IA-64), so that users on platforms with smaller pages do not accidentally create files which might cause problems for IA-64 clients.

Although you could create files with a stripe size of 16 KB, this would be a poor choice. Practically, the smallest recommended stripe size is 512 KB, because Lustre tries to batch I/O into 512 KB chunks over the network. This is a good amount of data to transfer at once, and choosing a smaller stripe size may hinder the batching.

Generally, a good stripe size for sequential I/O using high-speed networks is between 1 MB and 4 MB. Stripe sizes of larger than 4 MB will not parallelize as effectively, because Lustre tries to keep the amount of dirty cached data below 4 MB per server with the default configuration.

Writes which cross an object boundary are slightly less efficient than writes which go entirely to one server. Depending on your application's write patterns, you can assist it by choosing the stripe size with that in mind. If the file is written in a very consistent and aligned way, you can do it a favor by making the stripe size a multiple of the write() size.

The choice of stripe size has no effect on a single-stripe file.

2.2 Displaying Striping Information with lfs getstripe

Individual files and directories can be examined with *lfs getstripe*:

```
lfs getstripe <filename>
```

lfs will print the index and UUID for each OST in the file system, along with the OST index and object ID for each stripe in the file. For directories, the default settings for files created in that directory will be printed.

A whole tree of files can also be inspected with lfs find:

```
lfs find [--recursive | -r] <file or directory> ...
```

2.3 lfs setstripe – setting striping patterns

New files with a specific stripe configuration can be created with *lfs setstripe*:

```
lfs setstripe <filename> <stripe-size> <starting-ost>
<stripe-count>
```

If you pass a stripe-size of 0, the file system default stripe size will be used. Otherwise, the stripe-size must be a multiple of 16 KB.

If you pass a starting-ost of -1, a random first OST will be chosen. Otherwise the file will start on the specified OST index (starting at zero).

If you pass a stripe-count of 0, the file system default number of OSTs will be used. A stripe-count of -1 means that all available OSTs should be used.

2.3.1 Changing Striping for a Subdirectory

lfs setstripe works on directories to set a default striping configuration for files created within that directory. The usage is the same as for lfs setstripe for a regular file, except that the directory must exist prior to setting the default striping configuration on it. If a file is created in a directory with a default stripe configuration (without otherwise specifying the striping) Lustre will use those striping parameters instead of the file system default for the new file.

To change the striping pattern for a subdirectory, create a directory with desired striping pattern as described above. The subdirectories inherit the striping pattern of the parent directory.

2.3.2 Using a Specific Striping Pattern for a Single File

lfs setstripe will create a file with a given stripe pattern.

lfs setstripe will fail if the file already exists.

2.4 Performing Direct I/O

For more information about the pros and cons of using Direct I/O with Lustre, see *Performance Concepts.*

Pass the O_DIRECT flag to open()

Applications must use read() and write() calls only with buffers aligned on a 512-byte boundary; the size of the buffer must be a multiple of 512 bytes. If not, -EINVAL

2.4.1 Making File System Objects Immutable

An *immutable* file or directory is one that cannot be modified, renamed, or removed.

chattr +i <file>

chattr –i removes the flag

2.4.2 Disabling Locking for Direct I/O Accesses

Chapter IV – 3. Lustre Security

3.1 Using Access Control Lists

ACL, short for access control list, is a set of data that informs an operating system about the permissions, or access rights, that each user or group has to a specific system object, such as a directory or file. Each object has a unique security attribute that identifies which users have access to it, and the ACL is a list of each object and user access privileges such as read, write or execute.

3.1.1 How Do ACLs Work?

Implementation of the Access Control List (ACL) is varies with various operating systems. Systems that support the POSIX (Portable Operating System Interface) family of standards share a simple yet powerful file system permission model: Every file system object is associated with three sets of permissions that define access for the owner, the owning group, and for others. Each set may contain Read (r), Write (w), and Execute (x) permissions. This scheme is implemented using only nine bits. The Set User Id, Set Group Id, and Sticky bits are used for a number of special cases.

This model is extremely simple. It is sufficient for implementing the permission scenarios that usually occur on UNIX systems. Only the root user can create groups or change group memberships. Set-user-ID root utilities may allow ordinary users to perform some administrative tasks. This has eventually resulted in a number of Access Control List (ACL) implementations on UNIX, which are only compatible among each other to a limited degree.

3.1.2 Lustre ACLs

Starting with version 1.4.6 all versions of Lustre will support POSIX ACLs. When using a lustre client of version 1.4.5 or below with an MDS of version 1.4.6, or vice versa, the user space program generates an error “Operation Not Supported” during ACL operations.

MDS needs to be configured in order to use the ACL. While setting up the MDS, an option “ACL” is to be added as a parameter into the backup file system mount options.

```
lconf --node mds_node {--mountfsoptions=acl|--acl}
config.xml
```

Clients also need to be configured in order to use ACLs. When setting up the Clients, an option “ACL” is to be added as a parameter into the clients options.

```
lconf --node client_node {--clientoptions=acl|--acl}
OR config.xml
```

To determine if the luster file system is mounted with the ACL options, check the /etc/mtab file.

When the following command is executed,

```
cat /mnt/lustre
```

an entry will get added at the end.

```
uml /mnt/luster luster_lite rw, osc=Lov1,
mdc=MDC_client1.spsoftware.com_mds1_MNT_client, acl 0
0
```

Even when an MDS is setup without the parameter “ACL”, the client will get the error “Operation Not Supported” during ACL operations. The ACL options for a client and an MDS must match for security reasons. To use or not to use the ACL is determined by the MDS. If in case, the ACL gets disabled somehow and the client is trying to get/retrieve the ACLs, there will be no RPC (Remote Procedure Call) error generated on the network. It will only be the client user space getting the message of “Operation not Supported”. Any how the OSS will not receive any RPC error for the ACL Operations.

> NOTE: If the setup of the MDS is done with the “ACL” parameters, all the clients of version 1.4.5 or less than that will not be able to connect to the MDS as the ACLs do not match.

All the Lustre ACL operations are similar to any other file systems. Once the client and MDS conform to the correct requirements and agree to use the ACLs, the client will be able to manipulate the ACLs by standard utilities. The ACL package contains the libacl, getfacl and setfacl utilities. setfacl is used to set the Access Control Lists (ACLs) for files and directories. Whereas in the same way getfacl is used to get the Access Control Lists (ACLs) for files and directories. If setfacl is used on a file system which does not support ACLs, setfacl operates on the file mode permission bits. If the ACL does not fit completely in the permission bits, setfacl modifies the file mode permission bits to reflect the ACL as closely as possible. If getfacl is used on a file system that does not support ACLs, getfacl displays the access permissions defined by the traditional file mode permission bits.

Lustre ACL is an extension of traditional user/ group /other permissions. So, it is required that the lustre client and the MDS must share the same user database, which is usually achieved by LDAP, NIS, or by copying /etc/passwd and /etc/group all over. In Lustre, we call these clients as “local realm clients” or “local clients”.

> NOTE: All the above description is for the client and server of version 1.4.6 and above.

Chapter IV – 4. Other Lustre Operating Tips

4.1 Expanding the File System by Adding OSTs

It is not an ideal behavior when OSTs are full. However there are several roadmap items to address this issue.

The current scenario can be described as follows -

Once an OST has filled up, an application trying to write new data to it will receive an -ENOSPC error. This produces the unfortunate confusing behavior of allowing writes for some files (or even some parts of a striped file) while it denies writes for others. This is not an issue for newly-created files, as Lustre will avoid placing new files on full OSTs.

If empty OSTs are then added to the system, the system will be quite out of balance. Although Lustre does not yet have an on-line data migration function, you can re-balance your data manually.

Instructions for adding OSTs to existing Lustre file systems -

Step 1: Stop any existing Lustre clients and servers

Step 2: Modify or re-create the Lustre XML configuration (using lmc, as usual). You can do this before your stop your servers, as long as you save a copy of the old XML.

Add "lmc --add ost" commands to your lmc script.

Step 3: Run "lconf --write_conf" on the MDS.

This compiles the XML into a binary configuration log that is stored in the MDS. This log is processed by clients at mount-time, to allow them to mount without needing lconf or the XML.

Just add "--write_conf" to your normal MDS lconf startup command. It will start the MDS, compile the XML, and stop the MDS. If you skip this step, your clients will use the old configuration, with unpredictable results.

Step 4: Start your servers and clients

It is very important that these additional commands are *after* all of the current OSTs in the script. You will also need to reformat the OST device manually after --write_conf, at least in Lustre 1.4.6:

```
# mke2fs -j -J size=400 -I 256 -i 16384 /dev/DEVICE
```

The basic method is to copy existing files to a temporary file, then 'mv' the temp file over the old one -- naturally, this should not be attempted with files which are currently being written to by users or applications. This operation will redistribute the stripes over the entire set of OSTs. A sample script for this migration is attached.

A very clever migration script would:

- Examine the current distribution of data

- Calculate how much data should move from each full OST to the empty ones

- Search for files on a given full OST (using "lfs getstripe")

- Force the new destination OST (using lfs setstripe)

- Copy only enough files to address the imbalance

If an enterprising Lustre administrator wants to explore this approach, per-OST disk-usage statistics can be found under /proc/fs/lustre/osc/*

Future Development:

In the short term (4Q05-1Q06) Lustre will include a runtime option that will create proportionally more new files on OSTs with more room available. Although this won't help if you need to write new data to an existing file on a completely full OST, it will help to keep a system from getting too far out of balance in the first place, and help bring it back into balance more quickly.

The problem is best solved with a proper on-line data migrator, which can safely migrate even those files being actively modified. This is a very involved task, one which will likely not be completed in the coming year.

Part V. Reference

Chapter V – 1. User Utilities (man1)

1.1 lfs

lfs is a Lustre client file system utility that is used to create a file with a specific striping pattern, to find the striping pattern of an exiting file, and to set or create quotas.

1.1.1 Synopsis

```
lfs

lfs df

lfs find [--obd <uuid>] [--quiet | --verbose] [-- \
recursive] <dir|file>

lfs getstripe <file_name>

lfs setstripe <filename> <stripe_size> <start_ost>  \
<stripe_count>

lfs quotachown [ -i ] <filesystem>

lfs quotacheck [ -ugf ] <filesystem>

lfs quotaoff [-ug] <filesystem>

lfs setquota [-u | -g] <name> <block-softlimit> \
<block-hardlimit> <inode-softlimit> <inode-hardlimit>
\  <filesystem>

lfs quota [-o obd_uiid] [-u | -g] <name> <filesystem>

lfs check <mds| osts| servers>
```

NOTE: For the above example <filesystem> refers to the mount point of the Lustre file system (Default: /mnt/lustre).

1.1.2 Description

This utility is used to create a new file with a specific striping pattern, to determine the default striping pattern, to gather the extended attributes (object numbers and location) for a specific file and for setting lustre quota. It can be invoked interactively without any arguments or in a non-interactive mode.

lfs can be invoked in interactive mode by issuing the commands given below.

```
$ lfs

lfs> help
```

To get a complete listing of available commands, type help on the lfs prompt. To get basic help on meaning and syntax of a command, type help command. Command completion is activated with the TAB key, and command history is available via the UP and DOWN ARROWS.

setstripe Creates a new file with a specific striping pattern; or sets the default striping pattern on an existing directory; or deletes the default striping pattern from an existing directory

getstripe Lists the striping pattern for a given file name

find Lists the extended attributes for a given filename or files in a directory or recursively for all files in a directory tree. It can also be used to list the files that have objects on a specific OST.

df To check the disk space or inodes for each available MDS and OSTs

quotachown Changes the owner or group of a file on the specified file system

quotacheck Scans the specified file system for disk usage, and creates or updates quota files

quotaon Turns file system quotas on

quotaoff Turns file system quotas off

setquota Turns file system quotas on a file system

quota Displays the disk usage and limits

check Turns file system quotas on. Displays the status of MDS or OSTs (as specified in the command) or all the servers (MDS and OSTs)

osts Lists all the OSTs for the file system

help Provides brief help on various arguments

exit/quit Quits the interactive lfs session

1.1.3 Examples

To create a file striped on one OST (default is 1 MB):

```
$ setstripe <filename|dirname> <stripe size> \
<stripe start> <stripe count>
```

Or

```
$  setstripe -d <dirname>  (to delete default
striping)
```

stripe size: If you pass a stripe-size of 0, the file system default stripe size will be used. Otherwise, the stripe-size must be a multiple of 16 KB.

stripe start: If you pass a starting-ost of -1, a random first OST will be chosen. Otherwise the file will start on the specified OST index (starting at zero).

stripe count: If you pass a stripe-count of 0, the file system default number of OSTs will be used. A stripe-count of -1 means that all available OSTs should be used.

Below is an example of setting and getting stripes:

```
$ lfs > setstripe lustre.iso 0 -1 0

$ lfs > getstripe lustre.iso
```

```
OBDS:
0: ost1_UUID ACTIVE
1: ost2_UUID_2 ACTIVE
./lustre
    obdidx  objid  objid  group
      1       4     0x4     0
```

To list the extended attributes of a given file or directory:

```
$ find [--obd <uuid>] [--quiet | --verbose] \
[--recursive] <dir|file>
```

For instance,

```
$ lfs find /mnt/lustre/{filename|directory}
```

To recursively list the extended attributes of all the files in a given directory tree:

```
$ lfs find -r /mnt/lustre/
```

To list all the files that have objects on a specific OST:

```
$ lfs find -r --obd OST2-UUID /mnt/lustre/
```

To change the file owner and group:

```
$ lfs quotachown -i /mnt/lustre
```

To check the quota for a user and a group:

```
$ lfs quotacheck -ug /mnt/lustre
```

To turn on the quotas for a user and a group:

```
$ lfs quotaon -ug /mnt/lustre
```

To turn off the quotas for a user and a group:

```
$ lfs quotaoff -ug /mnt/lustre
```

To set the quotas for a user as 1GB block quota and 10,000 file quota:

```
$ lfs setquota -u {username} 0 1000000 0 10000 \
/mnt/lustre
```

To change the owner or group:

```
$ quotachown [-i] <filesystem>
```

-i: To ignore the error if the file does not exist.

For instance,

```
$ lfs quotachown -i {file|directory} /mnt/lustre
```

To check the disk space in inodes available on individual MDS and OST:

```
$ lfs df -i /mnt/lustre
```

UUID	Inodes	Used	Free	Use%	Mounted on
mds-1_UUID	53265600	28266	53237334	0	/mnt/lustre[MDT:0]
ost-1_UUID	244056064	1349	244054715	0	/mnt/lustre[OST:0]
ost-2_UUID	244056064	884	244055180	0	/mnt/lustre[OST:1]

To check the disk space in size available on individual MDS and OST:

```
$ lfs df -h /mnt/lustre
```

UUID	1K-blocks	Used	Available	Use%	Mounted on
mds-1_UUID	203.5M	12.1M	191.5M	5	/mnt/lustre[MDT:0]
ost-1_UUID	1.8G	384.7M	1.4G	20	/mnt/lustre[OST:0]
ost-2_UUID	1.8G	343.0M	1.5G	18	/mnt/lustre[OST:1]
ost-3_UUID	1.8G	332.2M	1.5G	18	/mnt/lustre[OST:2]

To list the quotas of a user:

```
$ lfs quota -u {username} /mnt/lustre
```

To check the status of all the servers – MDSs and OSTs

```
$ lfs check servers
```

To check the status of all the servers – MDSs

```
$ lfs check mds
```

To check the status of all the servers – OSTs

```
$ lfs check ost
```

To get the list of all the OSTs:

```
$ lfs osts
```

To list the logs of particular types:

```
$ lfs catinfo {keyword} [node name]
```

Keywords are one of the followings: config, deletions.

Node name must be provided when using the keyword config.

For instance,

```
$ lfs catinfo {config|dele*tions}{mdsnode|ostnode}
```

To join the files:

```
$ join <filename_A> <filename_B>
```

Chapter V – 2. Lustre Programming Interfaces (man3)

2.1 Introduction

This chapter describes the public programming interfaces for controlling various aspects of Lustre from userspace. These interfaces are generally not guaranteed to remain unchanged over time, although CFS will make an effort to notify the user community well in advance of major changes.

2.2 User/Group Cache Upcall

2.2.1 Name

/proc/fs/lustre/mds/mds-service/group_upcall – Look up a given user's group membership

2.2.2 Description

The *group_upcall* file contains the path to an executable that, when properly installed, is invoked to resolve a numeric UID to a group membership list. See *Using the User/Group Cache* for more information about this functionality. This utility should complete the mds_grp_downcall_data data structure (below) and write it to the */proc/fs/lustre/mds/mds-service/group_info* pseudo-file.

See *lustre/utils/l_getgroups.c* in the Lustre source distribution for an example upcall program.

2.2.3 Parameters

The name of the MDS service.

The numeric UID.

2.2.4 Data Structures

```
#include <lustre/lustre_user.h>

#define MDS_GRP_DOWNCALL_MAGIC 0x6d6dd620

struct mds_grp_downcall_data {
        __u32           mgd_magic;
        __u32           mgd_err;
        __u32           mgd_uid;
        __u32           mgd_gid;
        __u32           mgd_ngroups;
        __u32           mgd_groups[0];
};
```

Chapter V – 3. Config Files and Module Parameters (man5)

3.1 Introduction

LNET network hardware and routing are now configured via module parameters. Parameters should be specified in the /etc/modprobe.conf file, e.g.:

options lnet networks=tcp0,elan0

specifies that this node should use all the available tcp and elan interfaces.

Module parameters are read when the module is first loaded. Type-specific LND (Lustre Network Device) modules (e.g. ksocklnd) are loaded automatically by the **lnet** module when LNET starts (typically upon *modprobe ptlrpc*).

Under Linux 2.6, the LNET configuration parameters can be viewed under /sys/module/; generic and acceptor parameters under 'lnet' and LND-specific parameters under the corresponding LND's name.

Under Linux 2.4, sysfs is not available, but the LND-specific parameters are accessible via equivalent paths under /proc.

Important: All old Lustre configuration lines should be removed from the module configuration files, to be replaced with the following. Make sure that CONFIG_KMOD is set in your linux .config so that LNET can load the following modules it needs -

- modprobe.conf (Linux 2.6)
- options lnet networks=tcp0,elan0
- alias lustre llite
- modules.conf (Linux 2.4)
- options lnet networks=tcp0,elan0
- alias lustre llite

Default option settings are shown in parenthesis. Changes to parameters marked with a ‘W’ affect running systems. (Non-‘W’ can only be set when LNET loads for the first time.) Changes to parameters marked with a ‘Wc’ only have effect when connections are established (existing connections are not affected by these changes.)

3.2 Module Options

3.2.1 LNET Options

3.2.1.1 Network Topology

The network topology module parameters determine which networks a node should join, whether it should route between these networks and how it communicates with non-local networks.

> Note: Lustre will ignore the loopback interface (lo0). But Lustre will use any IP addresses aliased to the loopback by default. When in doubt, specify networks explicitly.

ip2nets ("") is a string that lists globally available networks, each with a set of IP address ranges. LNET determines the locally available networks from this list by matching the IP address ranges with the local IP's of a node. The purpose of this option is to be able to use the same modules.conf file across a variety of nodes on different networks. The string has the following syntax...

<ip2nets> :== <net-match> [<comment>] { <net-sep> <net-match> }

<net-match> :== [<w>] <net-spec> <w> <ip-range> { <w> <ip-range> }

[<w>]

<net-spec> :== <network> ["(" <interface-list> ")"]

<network> :== <nettype> [<number>]

<nettype> :== "tcp" | "elan" | "openib" | ...

<iface-list> :== <interface> ["," <iface-list>]

<ip-range> :== <r-expr> "." <r-expr> "." <r-expr> "." <r-expr>

<r-expr> :== <number> | "*" | "[" <r-list> "]"

<r-list> :== <range> ["," <r-list>]

<range> :== <number> ["-" <number> ["/" <number>]]

<comment :== "#" { <non-net-sep-chars> }

<net-sep> :== ";" | "\n"

<w> :== <whitespace-chars> { <whitespace-chars> }

The <net-spec> contains enough information to identify the network uniquely and load an appropriate LND. The LND determines the missing "address-within-network" part of the NID based on the interfaces it can use.

The optional <iface-list> specifies which hardware interface the network can use. If omitted, all the interfaces are used. LNDs that do not support the <iface-list> syntax cannot be configured to use particular interfaces and just use what is there. Only a single instance of these LNDs can exist on a node at any time, and the <iface-list> must be omitted.

The <net-match> entries are scanned in the order declared to see if one of the node's IP addresses matches one of the <ip-range> expressions. If there is a match, the <net-spec> specifies the network to instantiate. Note that it is the first match for a particular network that counts. This can be used to simplify the match expression for the general case by placing it after the special cases. For example..

ip2nets="tcp(eth1,eth2) 134.32.1.[4-10/2]; tcp(eth1) *.*.*.*" says that 4 nodes on the 134.32.1.* network have 2 interfaces (134.32.1.{4,6,8,10}) but all the rest have 1.

ip2nets="vib 192.168.0.*; tcp(eth2) 192.168.0.[1,7,4,12]" describes an IB cluster on 192.168.0.*. 4 of these nodes also have IP interfaces; these 4 could be used as routers.

Note that match-all expressions (e.g. *.*.*.*) effectively mask all other <net-match> entries specified after them; they should be used with caution.

3.2.1.2 networks ("tcp")

This is an alternative to "ip2nets" which can be used to specify the networks to be instantiated explicitly. The syntax is a simple comma separated list of <net-spec>s (see above). The default is only used if neither 'ip2nets' nor 'networks' is specified.

3.2.1.3 routes (“”)

This is a string that lists networks and the NIDs of routers that forward to them.

It has the following syntax (<w> is one or more whitespace characters):

<routes> :== <route>{ ; <route> }

<route> :== [<net>[<w><hopcount>]<w><nid>{<w><nid>}

So a node on the network tcp1 that needs to go through a router to get to the elan network

options lnet networks=tcp1 routes="elan 1 192.168.2.2@tcp1”

The hopcount is used to help choose the best path between multiply-routed configurations.

A simple but powerful expansion syntax is provided, both for target networks and router NIDs as follows...

<expansion> :== "[" <entry> { "," <entry> } "]"

<entry> :== <numeric range> | <non-numeric item>

<numeric range> :== <number> ["-" <number> ["/" <number>]]

The expansion is a list enclosed in square brackets. Numeric items in the list may be a single number, a contiguous range of numbers, or a strided range of numbers. For example, *routes="elan 192.168.1.[22-24]@tcp"* says that network elan0 is adjacent (hopcount defaults to 1); and is accessible via 3 routers on the tcp0 network (192.168.1.22@tcp, 192.168.1.23@tcp and 192.168.1.24@tcp).

routes="[tcp,vib] 2 [8-14/2]@elan" says that 2 networks (tcp0 and vib0) are accessible through 4 routers (8@elan, 10@elan, 12@elan and 14@elan). The hopcount of 2 means that traffic to both these networks will be traversed 2 routers - first one of the routers specified in this entry, then one more.

Duplicate entries, entries that route to a local network, and entries that specify routers on a non-local network are ignored.

Equivalent entries are resolved in favor of the route with the shorter hopcount. The hopcount, if omitted, defaults to 1 (i.e. the remote network is adjacent).

It is an error to specify routes to the same destination with routers on different local networks.

If the target network string contains no expansions, the hopcount defaults to 1 and may be omitted (i.e. the remote network is adjacent). In practice, this is true for most multi-network configurations. It is an error to specify an inconsistent hop count for a given target network. This is why an explicit hopcount is required if the target network string specifies more than one network.

3.2.1.4 forwarding ("")

This is a string that can be set either to "enabled" or "disabled" for explicit control of whether this node should act as a router, forwarding communications between all local networks.

A standalone router can be started by simply starting LNET ("*modprobe ptlrpc*") with appropriate network topology options

Acceptor

The acceptor is a TCP/IP service that some LNDs use to establish communications. If a local network requires it and it has not been disabled, the acceptor listens on a single port for connection requests that it redirects to the appropriate local network. The acceptor is part of the LNET module and configured by the following

accept

accept ("secure") is a string that can be set to any of the following values.

secure - accept connections only from reserved TCP ports (< 1023).

all - accept connections from any TCP port. Note: this is **required** for libLustre clients to allow connections on non-privledged ports.

none - do not run the acceptor

accept_port

accept_port (988) is the port number on which the acceptor should listen for connection requests. All nodes in a site configuration that require an acceptor must use the same port.

accept_backlog

accept_backlog (127) is the maximum length that the queue of pending connections may grow to (see listen(2)).

accept_timeout

accept_timeout (5,W) is the maximum time in seconds the acceptor is allowed to block while communicating with a peer.

accept_proto_version

accept_proto_version is the version of the acceptor protocol that should be used by outgoing connection requests. It defaults to the most recent acceptor protocol version, but it may be set to the previous version to allows the node to initiate connections with nodes that only understand that version of the acceptor protocol. The acceptor can, with some restrictions, handle either version (i.e. it can accept connections from both 'old' and 'new' peers). For the current version of the acceptor protocol (version 1), the acceptor is compatible with old peers if it is only required by a single local network.

3.2.2 SOCKLND Kernel TCP/IP LND

The socklnd is connection-based and uses the acceptor to establish communications via sockets with its peers.

It supports multiple instances and load balances dynamically over multiple interfaces. If no interfaces are specified by the *ip2nets* or *networks* module parameter, all non-loopback IP interfaces are used. The address-within-network is determined by the address of the first IP interface an instance of the socklnd encounters.

Consider a node on the “edge” of an Infiniband network, with a low bandwidth management ethernet (eth0), IP over IB configured (ipoib0), and a pair of GigE NICs (eth1,eth2) providing off-cluster connectivity. This node should be configured with "networks=vib,tcp(eth1,eth2)” to ensure that the socklnd ignores the management ethernet and IPoIB.

timeout (50,W) is the time in seconds that communications may be stalled before the LND will complete them with failure.

nconnds (4) sets the number of connection daemons.

min_reconnectms (1000,W) is the minimum connection retry interval in milliseconds. This sets the time that must elapse before the first retry after a failed connection attempt. As connections attempts fail, this time is doubled on each successive retry up to a maximum of 'max_reconnectms'.

max_reconnectms (60000,W) is the maximum connection retry interval in milliseconds.

eager_ack (0 on linux, 1 on darwin,W) is a boolean that determines whether the socklnd should attempt to flush sends on message boundaries.

typed_conns (1,Wc) is a boolean that determines whether the socklnd should use different sockets for different types of message. When clear, all communication with a particular peer takes place on the same socket. Otherwise separate sockets are used for bulk sends, bulk receives and everything else.

min_bulk (1024,W) determines when a message is considered "bulk".

buffer_size (8388608,Wc) sets the socket buffer size. Note that changes to this parameter may be rendered ineffective by other system-imposed limits (e.g. /proc/sys/net/core/wmem_max etc).

nagle (0,Wc) is a boolean that determines if nagle should be enabled. It should never be set in production systems.

keepalive_idle (30,Wc) is the time in seconds that a socket can remain idle before a keepalive probe is sent. 0 disables keepalives

keepalive_intvl (2,Wc) is the time in seconds to repeat unanswered keepalive probes. 0 disables keepalives.

keepalive_count (10,Wc) is the number of unanswered keepalive probes before pronouncing socket (hence peer) death.

irq_affinity (1,Wc) is a boolean that determines whether to enable IRQ affinity. When set, the socklnd attempts to maximize performance by handling device interrupts and data movement for particular (hardware) interfaces on particular CPUs. This option is not available on all platforms.

zc_min_frag (2048,W) determines the minimum message fragment that should be considered for zero-copy sends. Increasing it above the platform's PAGE_SIZE will disable all zero copy sends. This option is not available on all platforms.

3.2.3 QSW LND

The qswlnd is connectionless, therefore it does not need the acceptor.

It is limited to a single instance, which uses all Elan "rails" that are present and load balances dynamically over them.

The address-with-network is the node's Elan ID. A specific interface cannot be selected in the "networks" module parameter.

tx_maxcontig (1024) is a integer that specifies the maximum message payload in bytes to copy into a pre-mapped transmit buffer.

ntxmsgs (8) is the number of "normal" message descriptors for locally initiated communications that may block for memory (callers block when this pool is exhaused).

nnblk_txmsg (512 with a 4K page size, 256 otherwise) is the number of "reserved" message descriptors for communications that may not block for memory. This pool must be sized large enough so that it is never exhausted.

nrxmsg_small (256) is the number of "small" receive buffers to post (typically everything apart from bulk data).

ep_envelopes_small (2048) is the number of message envelopes to reserve for the "small" receive buffer queue. This determines a breakpoint in the number of concurrent senders. Below this number, communication attempts are queued, but above this number, the pre-allocated envelope queue will fill, causing senders to back off and retry. This can have the unfortunate side effect of starving arbitrary senders, who continually find the envelope queue is full when they retry. This parameter should therefore be increased if envelope queue overflow is suspected.

nrxmsg_large (64) is the number of "large" receive buffers to post (typically for routed bulk data).

ep_envelopes_large (256) is the number of message envelopes to reserve for the "large" receive buffer queue. See "ep_envelopes_small" above for a further description of message envelopes.

optimized_puts (32768,W) is the smallest non-routed PUT that will be RDMA-ed.

optimized_gets (1,W) is the smallest non-routed GET that will be RDMA-ed.

3.2.4 RapidArray LND

The ralnd is connection-based and uses the acceptor to establish connections with its peers.

It is limited to a single instance, which uses all (both) RapidArray devices present. It load balances over them using the XOR of the source and destination NIDs to determine which device to use for any communication.

The address-within-network is determined by the address of the single IP interface that may be specified by the "networks" module parameter. If this is omitted, the first non-loopback IP interface that is up is used instead.

n_connd (4) sets the number of connection daemons.

min_reconnect_interval (1,W) is the minimum connection retry interval in seconds. This sets the time that must elapse before the first retry after a failed connection attempt. As connections attempts fail, this time is doubled on each successive retry up to a maximum of 'max_reconnect_interval'.

max_reconnect_interval (60,W) is the maximum connection retry interval in seconds.

timeout (30,W) is the time in seconds that communications may be stalled before the LND will complete them with failure

ntx (64) is the number of "normal" message descriptors for locally initiated communications that may block for memory (callers block when this pool is exhaused).

ntx_nblk (256) is the number of "reserved" message descriptors for communications that may not block for memory. This pool must be sized large enough so that it is never exhausted.

fma_cq_size (8192) is the number of entries in the RapidArray FMA completion queue to allocate. It should be increased if the ralnd starts to issue warnings that the FMA CQ has overflowed. This is only a performance issue.

max_immediate (2048,W) is the size in bytes of the smallest message that will be RDMA-ed, rather than being included as immediate data in an FMA. All messages over 6912 bytes must be RDMA-ed (FMA limit).

3.2.5 VIB LND

The vib lnd is connection based, establishing reliable queue-pairs over Infiniband with its peers. It does not use the acceptor for this.

It is limited to a single instance, which uses a single HCA that can be specified via the "networks" module parameter. It this is omitted, it uses the first HCA in numerical order it can open.

The address-within-network is determined by the IPoIB interface corresponding to the HCA used.

service_number (0x11b9a2) is the fixed IB service number on which the LND listens for incoming connection requests. Note that all instances of the viblnd on the same network must have the same setting for this parameter.

arp_retries (3,W) is the number of times the LND will retry ARP while it establishes communications with a peer.

min_reconnect_interval (1,W) is the minimum connection retry interval in seconds. This sets the time that must elapse before the first retry after a failed connection attempt. As connections attempts fail, this time is doubled on each successive retry up to a maximum of 'max_reconnect_interval'.

max_reconnect_interval (60,W) is the maximum connection retry interval in seconds.

timeout (50,W) is the time in seconds that communications may be stalled before the LND will complete them with failure.

ntx (32) is the number of "normal" message descriptors for locally initiated communications that may block for memory (callers block when this pool is exhaused).

ntx_nblk (256) is the number of "reserved" message descriptors for communications that may not block for memory. This pool must be sized large enough so that it is never exhausted.

concurrent_peers (1152) is the maximum number of queue pairs, and therefore the maximum number of peers that the instance of the LND may communicate with.

hca_basename ("InfiniHost") is used to construct HCA device names by appending the device number.

ipif_basename ("ipoib") is used to construct IPoIB interface names by appending the same device number as is used to generate the HCA device name.

local_ack_timeout (0x12,Wc) is a low-level QP parameter. It should not be changed from the default unless advised.

retry_cnt (7,Wc) is a low-level QP parameter. It should not be changed from the default unless advised.

rnr_cnt (6,Wc) is a low-level QP parameter. It should not be changed from the default unless advised.

rnr_nak_timer (0x10,Wc) is a low-level QP parameter. It should not be changed from the default unless advised.

fmr_remaps (1000) controls how often FMR mappings may be reused before they must be unmapped. It should not be changed from the default unless advised.

cksum (0,W) is a boolean that determines whether messages (NB not RDMAs) should be checksummed. This is a diagnostic feature that should not be enabled normally.

3.2.6 OpenIB LND

The openib lnd is connection based and uses the acceptor to establish reliable queue-pairs over infiniband with its peers.

It is limited to a single instance that uses only IB device '0'.

The address-within-network is determined by the address of the single IP interface that may be specified by the "networks" module parameter. If this is omitted, the first non-loopback IP interface that is up, is used instead. It uses the acceptor to establish connections with its peers.

n_connd (4) sets the number of connection daemons. The default is 4.

min_reconnect_interval (1,W) is the minimum connection retry interval in seconds. This sets the time that must elapse before the first retry after a failed connection attempt. As connections attempts fail, this time is doubled on each successive retry up to a maximum of 'max_reconnect_interval'.

max_reconnect_interval (60,W) is the maximum connection retry interval in seconds.

timeout (50,W) is the time in seconds that communications may be stalled before the LND will complete them with failure.

ntx (64) is the number of "normal" message descriptors for locally initiated communications that may block for memory (callers block when this pool is exhausted).

ntx_nblk (256) is the number of "reserved" message descriptors for communications that may not block for memory. This pool must be sized large enough so that it is never exhausted.

concurrent_peers (1024) is the maximum number of queue pairs, and therefore the maximum number of peers that the instance of the LND may communicate with.

cksum (0,W) is a boolean that determines whether messages (NB not RDMAs) should be checksummed. This is a diagnostic feature that should not be enabled normally.

3.2.7 Portals LND (Linux)

The ptllnd can be used as a interface layer to communicate with Sandia Portals networking devices. This version is intended to work on the Cray XT3 Linux nodes using Cray Portals as a network transport.

Message Buffers - When ptllnd starts up, it allocates and posts sufficient message buffers to allow all expected peers (set by 'concurrent_peers') to send 1 message unsolicited. The first message a peer actually sends is a (so-called) "HELLO" message, which is used to negotiate how much additional buffering to set up; typically 8 messages. So if 10000 peers actually exist, we will post enough buffers for 80000 messages.

The maximum message size is set by the 'max_msg_size' module parameter (default 512). This parameter sets the bulk transfer breakpoint. Below this breakpoint, payload data is sent in the message itself, and above this breakpoint, a buffer descriptor is sent and the receiver gets the actual payload.

The buffer size is set by the 'rxb_npages' module parameter (default 1). The default conservatively avoids allocation problems due to kernel memory fragmentation. However increasing this to 2 is probably not risky.

The ptllnd also keeps an additional 'rxb_nspare' buffers (default 8) posted to account for full buffers being handled.

Assuming a 4K page size, with 10000 peers, 1258 buffers can be expected to be posted at startup, rising to a max of 10008 as peers actually connected. This could be reduced by a factor of 4 by doubling rxb_npages halving 'max_msg_size'.

ME/MD queue length - The ptllnd uses a single portal set by the 'portal' module parameter (default 9) for both message and bulk buffers. Message buffers are always attached with PTL_INS_AFTER and match anything sent with "message" matchbits. Bulk buffers are always attached with PTL_INS_BEFORE and match only specific matchbits for that particular bulk transfer.

This scheme assumes that the majority of ME/MDs posted are for "message" buffers, and that the overhead of searching through the preceding "bulk" buffers is acceptable. Since the number of "bulk" buffers posted at any time is also dependent on the bulk transfer breakpoint set by 'max_msg_size', this seems like an issue worth measuring at scale.

TX descriptors - The ptllnd has a pool of so-called "tx descriptors", which it uses not only for outgoing messages, but also to hold state for bulk transfers requested by incoming messages. This pool should therefore scale with the total number of peers.

To enable the building of the Portals LND (ptllnd.ko) configure with the following option:

./configure --with-portals=<path-to-portals-headers>

ntx (256) The total number of message descriptors

concurrent_peers (1152) The maximum number of concurrent peers. Peers attempting to connect beyond the maximum will not be allowd.

peer_hash_table_size (101) The number of hash table slots for the peers. This number should scale with concurrent_peers. The size of the peer hash table is set by the module parameter 'peer_hash_table_size' which defaults 101. This number should be prime to ensure the peer hash table is populated evenly. Increasing this to 1001 for~10000 peers is advisable.

cksum (0) Set to non-zero to enable message (not RDMA) checksums for outgoing packets. Incoming packets will always be checksumed if necssary, independnt of this value.

timeout (50) The amount of time a request can linger in a peers active queue, before the peer is considered dead. Units: seconds.

portal (9) The portal ID to use for the ptllnd traffic.

rxb_npages (64 * #cpus) The number of pages in a RX Buffer.

credits (128) The maximum total number of concurrent sends that are outstanding at any given instant.

peercredits (8) The maximum number of concurrent sends that are outstanding to a single peer at any given instant.

max_msg_size (512) The maximum immedate message size. This MUST be the same on all nodes in a cluster. A peer connecting with a diffrent max_msg_size will be rejected.

Portals LND (Catamount)

The ptllnd can be used as a interface layer to communicate with Sandia Portals networking devices. This version is intended to work on the Cray XT3 Catamount nodes using Cray Portals as a network transport.

To enable the building of the Portals LND configure with the following option:

./configure --with-portals=<path-to-portals-headers>

The following environment variables can be set to configure the PTLLND's behavior.

PTLLND_PORTAL (9) The portal ID to use for the ptllnd traffic.

PTLLND_PID (9) The virtual pid on which to contact servers.

PTLLND_PEERCREDITS (8) The maximum number of concurrent sends that are outstanding to a single peer at any given instant.

PTLLND_MAX_MESSAGE_SIZE (512) The maximum messages size. This MUST be the same on all nodes in a cluster.

PTLLND_MAX_MSGS_PER_BUFFER (64) The number of messages in a receive buffer. Receive buffer will be allocated of size PTLLND_MAX_MSGS_PER_BUFFER times PTLLND_MAX_MESSAGE_SIZE.

PTLND_MSG_SPARE (256) Additional receive buffers posted to portals.

PTLLND_PEER_HASH_SIZE (101) The number of hash table slots for the peers.

PTLLND_EQ_SIZE (1024) The size of the Portals event queue (i.e. maximum number of events in the queue).

Chapter V – 4. System Configuration Utilities (man8)

4.1 lmc

lmc (Lustre configuration maker) is invoked for generating configuration data files (.xml).

4.1.1 Synopsis

```
lmc [options] --add <objecttype> [args]
```

4.1.2 Description

When invoked, ***lmc*** adds configuration data to config files (.xml). This data is regarding all the components of a Lustre cluster, like MDSs, mount-points, OSTs, LOVs and others. A single configuration file would be generated for the entire cluster. In the ***lmc*** command line interface, each of these components is associated with the *objecttype*.

The *objecttype* can be anyone of the following - **net, MDS, LOV, OST, mtpt, route, echo_client, or cobd.** Every *objecttype* refers to a collection of related configuration entities.

Following options can be used to generate the configuration data associated with all the *objecttypes* in a Lustre cluster:

-v, --verbose Prints system commands as they run

-m, --merge Appends to the specified config file

-o, --output Writes xml configuration into given output file (Overwrites the existing one)

-i, --input Takes input from a specified file

--batch Used to execute the LMC command in batch mode

Node related options:

--add node Adds a new node in the cluster configuration

> **--node {nodename}** Creates a new node with the given name if not already present
>
> **--timeout <num>** Timeout before going into recovery
>
> **--lustre_upcall <path>** Sets the location of both the upcall scripts (Lustre and Portals) used by the client for recovery
>
> **--portals_upcall <path>** Specifies the location of the Portals upcall scripts used by the client for recovery
>
> **--upcall <path>** Specifies the location of both the upcall scripts (Lustre and Portals) used by the client for recovery
>
> **--group_upcall <path>** Specifies the location of the group upcall scripts used by the MDS for determining supplementary group membership

--ptldebug <debug_level> Sets the portals debug level

--subsystem <subsystem_name> Specifies the Lustre subsystems, which have debug output recorded in the log

Network related options:

--add net Adds a network device descriptor for the given node

--node {node name} Creates a new node with the given name if not already present. This is also used to specify a specific node for other elements.

--nettype <type> This can be tcp, elan, or gm

--nid <nid> The network id, e.g. ElanID or IP address as used by Portals. If NID is "*", then the local address of the interface with specified nettype will be substituted when the node is configured with lconf. An NID with "*" should be used only for generic client configurations.

--cluster_id <id> Specifies the cluster ID

--hostaddr <addr> Specifies the host address, which will be transfered to the real host address by lconf

--router Optional flag to mark this node as a router

--port [port] Optional argument to indicate the tcp port. The default is 988.

--tcpbuf <size> Optional argument. The default TCP buffer size is 1MB.

--irq_affinity 0 or 1 Optional argument. Default is 0.

--nid_exchange 0 or 1 Optional argument as some of the OSTs might not have the required support. This is turned off by default, value of 1 will turn it ON.

MDS related options:

--add mds Specifies the MDS configuration

--node <node name> Name of the node on which the MDS resides

--mds <mds_name> Host name of the MDS

--mdsuuid <uuid> Specifies MDS UUID

--dev <pathname> Path of the device on the local system. If the device is a file, then a loop device is created and used as a block device.

--backdev <pathname> Path of the device for backing storage on the local system

--size <size> Optional argument indicating the size (in KB) of the device to be created (used typically for loop devices)

--node {nodename} Adds an MDS to the specified node. This requires a --node argument, and it should not be a profile node.

--fstype ldiskfs|ext3 Optional argument used to specify the file system type. Default is ext3. For 2.6 kernels, the ldiskfs file system must be used.

--inode_size <size> Specifies new inode size for underlying ext3 file system. Must be a power of 2 between 128 and 4096. The default inode size is selected based on the default number of stripes specified for the file system.

--group_upcall <pathname> The group upcall program used to resolve a user as a secondary group. The default value is NONE, which means that the MDS will use whatever supplementary group is passed from the client. The supplied upcall is /usr/sbin/l_getgroups, which gets groups from the MDS as /etc/group file based on the client-supplied UID.

--mkfsoptions <options> Optional argument to mkfs

--mountfsoptions <options> Optional argument to mountfs. Mount options will be passed by this argument. For example, extents are to be enabled by adding ",extents" to the option --mountfsoptions. Also, the option "asyncdel" can be added to it.

--journal_size <size> Optional argument to specify the journal size for the ext3 file system. The size should be in the units expected by mkfs. For ext3, it should be in MB. If this option is not used, the ext3 file system will be configured with the default journal size.

LOV related options:

--add lov Creates an LOV with the specified parameters. The mds_name must already exist in the descriptor.

--lov <name> Common name for the LOV

--mds <name> Host name for the MDS

--stripe_sz <size> Specifies the stripe size in bytes. Data exactly equal to this size is written in each stripe before starting to write in the next stripe. Default is 1048576.

--stripe_cnt <count> A value of 0 for this means Lustre is using the currently optimal number of stripes. Default is 1 stripe per file.

--stripe_pattern <pattern> Only Pattern 0 (RAID 0) is supported currently

OST related options:

--add os Creates an OBD, OST, and OSC. The OST and OBD are created on the specified node.

--ost <name> Assigns a name to the OST

--node <nodename> Node on which the OST service is running. It must not be a profile node.

--failout Disables failover support on the OST

--failover Enables failover support on the OST

--dev <pathname> Path of the device on the local system. If the device is a file, then a loop device is created and used as a block device.

--size [size] Optional argument indicating the size (in KB) of the device to be created (used typically for loop devices)

--obdtype <Obdfilter | obdecho> Specifies the type of the OSD

--lov <name> Name of the LOV to which this OSC will be attached. This is an optional argument.

--ostuuid UUID Specifies the UUID of the OST device

--fstype <extN | ext3> Optional argument used to specify the file system type. Default is ext3.

--inode_size <size> Specifies the new inode size for underlying ext3 file system

--mkfsoptions <options> Specifies additional options for the mkfs command line.

--mountfsoptions <options> Specifies additional options for mountfs command line. Mount options will be passed by this argument. For example, extents are to be enabled by adding ",extents" to the option --mountfsoptions. Also, the option "asyncdel" can be added to it.

--journal_size <size> Optional argument to specify the journal size for the ext3 file system. The size should be in the units expected by mkfs. For ext3, it should be in MB. If this option is not used, the ext3 file system will be configured with the default journal size.

Mountpoint related options:

--add mtpt Creates a mount-point on the specified node. Either an LOV name or an OSC name can be used.

--node {nodename} Node that will use the mountpoint

--path /mnt/path The mountpoint that can be used to mount Lustre file system. Default is /mnt/lustre.

--ost ost_name or **--lov lov_name** The OST or LOV name as specified earlier in the configuration

Route related options:

--add route Creates a static route through a gateway to a specific NID or a range of NIDs

--node {nodename} The node on which the route can be added

--router Optional flag to mark a node as the router

--gw nid The NID of the gateway. It must be a local interface or a peer.

--gateway_cluster_id id Specifies the id of the cluster, to which the gateway belongs

--target_cluster_id id Specifies the id of the cluster, to which the target of the route belongs

--lo nid The low value NID for a range route

--hi nid The high value NID for a range route

--add echo The client used exclusively for testing purpose

4.1.3 Examples

```
$ lmc --node adev3 --add net --nid adev3 --cluster_id
0x1000 --nettype tcp --hostaddr adev3-eth0 --port 988
```

```
$ lmc --node adev3 --add net --nid adev3 --cluster_id
0x2000 --nettype tcp --hostaddr adev3-eth1 --port 989
```

These commands are used to add a Lustre node to the specified Lustre cluster through a network interface. In this example, Lustre node adev3 has been added to 2 Lustre clusters separately through 2 network interface cards: adev3-eth0 and adev3-eth1. The cluster_ids of these 2 Lustres are 0x1000 and 0x2000. adev3 would listen to the respective specified port(s) to prepare for possible connection requests from nodes in these two clusters.

```
$ lmc --node adev3 --add route --nettype tcp --gw 5
--gateway_cluster_id 0x1000 --target_cluster_id
0x1000 --lo 4 --hi 7
```

This command is used to add a route entry for a Lustre node. Here Lustre node adev3 is added with a new route entry. This enables the Lustre node to send packets to Lustre nodes having the NIDs from 4 to 7 with the help of Lustre gateway node having the NID 5. Besides, Lustre gateway node is in the cluster of id 0x1000 and target of the route belongs to the cluster of the same id 0x1000. The network in this route path is a tcp network.

> NOTE:- When using --mountfsoptions {extents|mballoc|asyncdel}, please remember the following:
> -extents and mballoc are recommended only for 2.6 kernel and are used only for OSTs.
> -asyncdel is recommended for 2.4 kernel and is not supported on 2.6 kernel.
> One can use --mountfsoptions {extents|mballoc} on existing file systems. The Lustre servers need to be restarted before using this command so that the new options become effective.
> asyncdel is used on 2.4 kernel to delete files in a separate thread. Using this option quickly releases inode semaphore of the parent directory in order to perform the other operations. Otherwise, deleting large files may take more time. For 2.6 kernel, this is not such a big issue because the parameter extents can increase the speed of deletion.

4.2 lconf

lconf is a Lustre utility that is used for configuring, starting and stopping a Lustre file system.

4.2.1 Synopsis

```
lconf [OPTIONS] <XML-config file>
```

4.2.2 Description

This utility configures a node by using the configuration data given in the <XML-config-file>. For all the nodes in a cluster, there is one single configuration file. Thus, this file should be distributed to all the nodes in the cluster or kept at a location accessible to all the nodes. The options that can be used with ***lconf*** are explained below. The XML file must be specified. When invoked with no options, lconf will attempt to configure the resources owned by the node it is invoked on.

--abort_recovery Used to start Lustre when recovery is certain to fail (for example when an OST is disabled). Can also be used to skip the recovery timeout period.

--acl Enables Access Control List support on the client.

--allow_unprivileged_port Allows connections from unprivileged ports.

--clientoptions Additional options for mounting a Lustre client – *This is obsolete now and is replaced by zeroconfig mounts.*

--client_uuid The failed client (required for recovery).

--clumanager Generates a Red Hat Clumanager configuration file for this node.

--config The cluster configuration name used for LDAP query (depreciated).

--conn_uuid The failed connection (required for recovery).

-d --clenaup Stops Lustre and unconfigures a node. The same config and --node argument used for configuration needs to be used for cleanup as well. This will attempt to undo all of the configuration steps done by lconf, including unloading the kernel modules.

--debug_path <path> Specifies the path for saving debug dumps. (Default is /tmp/lustre-log.)

--dump <file> Dumps the kernel debug log to the specified file before portals is unloaded during clean up.

--failover Shuts down without saving state. This allows a node to give up service to another node for failover purposes. This is not a clean shutdown.

-f --force Forces unmounting and/or obd detach during cleanup.

--gdb Creates a gdb module script before doing any Lustre configuration.

--gdb_script Full name of the gdb debug script. Default is /tmp/ogdb.

--group The group of devices to cleanup/ configure.

--group_upcall The group upcall program used to resolve a user as a secondary group. The default value is NONE, which means that the MDS will use whatever supplementary group is passed from the client. But this is limited to a single supplementary group.

-h, --help Displays help

--inactive The UUID of the service to be ignored by the client which is mounting Lustre. It allows the client to mount in presence of some inactive services (currently OST only). Multiple UUIDs can be specified by repeating the option.

--lctl-dump Dumps all the ioctls to the specified file.

--ldapurl LDAP server URL (depreciated).

--lustre=<dir> The base directory of Lustre sources. This parameter causes lconf to load modules from a source tree.

--lustre_upcall Sets the location of the Lustre upcall scripts used by the client for recovery.

--make_service_scripts Creates per-service symlinks for use with clumanager HA software.

--maxlevel Performs configuration of devices and services up to the given level. When used in conjunction with cleanup, services are torn down up to a certain level.

Levels are approximately like:

10 – network

20 – device, ldlm

30 – OSD, MDD

40 – MDS, OST

70 – mountpoint, echo_client, OSC, MDC, LOV

--minlevel Specifies the minimum level of services to configure/ cleanup. Default is 0.

--mkfsoptions Specifies additional options for the mk*fs command line.

--mountfsoptions Specifies additional options for mountfs command line. Mount options will be passed by this argument. For example, extents are to be enabled by adding ",extents" to the option --mountfsoptions. Also, the option "asyncdel" can be added to it.

--node Specifies a specific node to be configured. By default, lconf searches for nodes with the local hostname and localhost. When --node is used, only node_name is searched for. If a matching node is not found in the config, then lconf exits with an error.

--noexec, -n Displays, but does not execute the steps to be performed by lconf. This is useful for debugging a configuration, and when used with --node, it can be run on any host.

--nomod Configures devices and services only. Does not load modules.

--nosetup Loads modules only. Does not configure devices or services.

--old_conf Starts up service even if config logs appear outdated.

--portals Specifies portals source directory. If this is a relative path, then it is assumed to be relative to Lustre. (Depreciated.)

--portals_upcall Specifies the location of the Portals upcall scripts used by the client for recovery. (Depreciated.)

--ptldebug Sets the portals debug level.

--quota Enables quota support for client file system.

--rawprimary <arg> For clumanager, device of the primary quorum. Default is /dev/raw/raw1.

--rawsecondary <arg> For clumanager, device of the secondary quorum. Default is /dev/raw/raw2.

--record Writes config information on the MDS.

--record_device Recovers a device.

--record_log Specifies the name of a config record log.

--recover Specifies the MDS device name that records the config commands.

--reformat Reformats all the devices. This is essential when the file system is brought up for first time.

--select Selects a particular node for a service

--service Shorthand for --group <arg> --select <arg>=<hostname>

--service_scripts <arg> For clumanager, directory containing per-service scripts. Default is /etc/lustre/services.

--single_socket The socknal option. Uses only one socket instead of a bundle.

--subsystem Sets the portals debug subsystem.

--tgt_uuid Specifies the failed target (required for recovery).

--timeout Sets the recovery timeout period.

--upcall Sets the location of both the upcall scripts (Lustre and Portals) used by the client for recovery.

--user_xattr Enables user_xattr support on the MDS.

--verbose, -v Becomes verbose and shows actions performed during the execution of a command

--write_conf Saves the whole client configuration information on the MDS

4.2.3 Examples

To invoke lconf on the OST node:

```
$ lconf -v --reformat --node ost config.xml
configuring for host:  ['ost']
setting /proc/sys/net/core/rmem_max to at least\
16777216
setting /proc/sys/net/core/wmem_max to at least\
16777216
Service: network NET_ost_tcp NET_ost_tcp_UUID
loading module: libcfs srcdir None devdir libcfs
+ /sbin/modprobe libcfs
loading module: lnet srcdir None devdir lnet
+ /sbin/modprobe lnet
+ /sbin/modprobe lnet
loading module: ksocklnd srcdir None devdir\
klnds/socklnd
+ /sbin/modprobe ksocklnd
Service: ldlm ldlm ldlm_UUID
loading module: lvfs srcdir None devdir lvfs
+ /sbin/modprobe lvfs
loading module: obdclass srcdir None devdir obdclass
+ /sbin/modprobe obdclass
loading module: ptlrpc srcdir None devdir ptlrpc
+ /sbin/modprobe ptlrpc
Service: osd OSD_ost1_ost OSD_ost1_ost_UUID
loading module: ost srcdir None devdir ost
+ /sbin/modprobe ost
loading module: ldiskfs srcdir None devdir ldiskfs
+ /sbin/modprobe ldiskfs
loading module: fsfilt_ldiskfs srcdir None devdir
lvfs
+ /sbin/modprobe fsfilt_ldiskfs
loading module: obdfilter srcdir None devdir
obdfilter
+ /sbin/modprobe obdfilter
+ sysctl lnet/debug_path /tmp/lustre-log-ost
+ /usr/sbin/lctl  modules > /tmp/ogdb-ost
Service: network NET_ost_tcp NET_ost_tcp_UUID
NETWORK: NET_ost_tcp NET_ost_tcp_UUID tcp ost
Service: ldlm ldlm ldlm_UUID
```

```
Service: osd OSD_ost1_ost OSD_ost1_ost_UUID
OSD: ost1 ost1_UUID obdfilter /lustre/filedevice/ost\
100000 ldiskfs no 0 0
+ losetup /dev/loop0
+ losetup /dev/loop1
+ losetup /dev/loop2
+ losetup /dev/loop3
+ losetup /dev/loop4
+ losetup /dev/loop5
+ losetup /dev/loop6
+ losetup /dev/loop7
+ dd if=/dev/zero bs=1k count=0 seek=100000\
of=/lustre/filedevice/ost
+ mkfs.ext2 -j -b 4096  -F
/lustre/filedevice/ost\  25000
+ tune2fs -O dir_index /lustre/filedevice/ost
+ losetup /dev/loop0
+ losetup /dev/loop0 /lustre/filedevice/ost
+ dumpe2fs -f -h /dev/loop0
no external journal found for /dev/loop0
OST mount options: errors=remount-ro
+ /usr/sbin/lctl
  attach obdfilter ost1 ost1_UUID
  quit
+ /usr/sbin/lctl
  cfg_device ost1
  setup /dev/loop0 ldiskfs f errors=remount-ro
  quit
+ /usr/sbin/lctl
  attach ost OSS OSS_UUID
  quit
+ /usr/sbin/lctl
  cfg_device OSS
  setup
  quit
```

To invoke lconf on the client node:

```
$ lconf --node client config.xml
```

To set the required debug levels:

```
$ lconf --ptldebug ~(portals | mballoc | trace)
```

To turn-on specific debug types:

```
$ conf --ptldebug ldlm|ha
```

A subset of failed OSTs can be ignored during Lustre mount on the clients by using the option below. Here OST1 and OST2 have failed and need to be ignored.

```
$ lconf --inactive OST_ost1_UUID --inactive
OST_ost2_UUID config.xml
```

To configure a node (the options in the square brackets are optional):

```
$ lconf --node {nodename} [--service name]] [--
reformat [--force [--failover]] [--reformat]  [--
mountfsoptions={options}] config.xml
```

> NOTE:- When using --mountfsoptions {extents|mballoc|asyncdel}, please remember the following:
> -extents and mballoc are recommended only for 2.6 kernel and are used only for OSTs.
> -asyncdel is recommended for 2.4 kernel and is not supported on 2.6 kernel.
> One can use --mountfsoptions {extents|mballoc} on existing file systems. The Lustre servers need to be restarted before using this command so that the new options become effective.
> asyncdel is used on 2.4 kernel to delete files in a separate thread. Using this option quickly releases inode semaphore of the parent directory in order to perform the other operations. Otherwise, deleting large files may take more time. For 2.6 kernel, this is not such a big issue because the parameter extents can increase the speed of deletion.

4.3 lctl

lctl is a Lustre utility used for Low level configurations of Lustre file system.

4.3.1 Synopsis

```
lctl

lctl --device <devno> <command [args]>

lctl --threads <numthreads> <verbose> <devno>
<command [args]>
```

4.3.2 Description

lctl can be invoked in interactive mode by issuing the commands given below.

```
$ lctl

lctl> help
```

The most common commands in lctl are in matching pairs - like device and attach, detach and setup, cleanup and connect, disconnect and help and quit. To get a complete listing of available commands, type help on the lctl prompt. To get basic help on meaning and syntax of a command, type help command. Command completion is activated with the TAB key, and command history is available via the “UP” and “DOWN” arrow keys.

For non-interactive single threaded use, one uses the second invocation, which runs command after connecting to the device.

Network related options:

--net <tcp/elan/myrinet> The network type to be used for the operation

network <tcp/elans/myrinet> Indicates what kind of network is applicable for the configuration commands that follow

interface_list Displays the interface entries

add_interface Adds an interface entry

del_interface [ip] Deletes an interface entry

peer_list Displays the peer entries

add_peer <nid> <host> <port> Adds a peer entry

del_peer [<nid>] [<host>] [ks] Removes a peer entry

conn_list Displays all the connected remote NIDs

connect [[<hostname> <port>] | <elan id>] Establishes connection to a remote network id given by the hostname/ port combination, or the elan id

disconnect <nid> Disconnects from a remote NID

active_tx Displays active transmits, and is used only for elan network type

mynid [nid] Informs the socknal of the local NID. It defaults to hostname for tcp networks, and is automatically setup for elan/ myranet networks

shownid Displays the local NID

add_uuid <uuid> <nid> Associates a given UUID with an NID

close_uuid <uuid> Disconnects a UUID

del_uuid <uuid> Deletes a UUID association

add_route <gateway> <target> [target] Adds an entry to the routing table for the given target

del_route <target> Deletes an entry for a target from the routing table

set_route <gateway> <up/down> [<time>] Enables/ disables routes via the given gateway in the protals routing table. <time> is used to specify when a gateway should come back online.

route_list Displays the complete routing table

fail nid|_all_ [count] Fails/ restores communications. Omitting the count implies an indefinite fail. A count of zero indicates that communication should be restored. A non-zero count indicates the number of LNET messages to be dropped after which the communication is restored. The argument "nid" is used to specify the gateway, which is one peer of the communication.

Device Selection:

newdev Creates a new device

device Selects the specified OBD device. All other commands depend on the device being set.

**cfg_device <$name>** Sets the current device being configured to <$name>

device_list Shows all the devices

lustre_build_version Displays the Lustre build version

Device Configuration:

attach type [name [uuid]] Attaches a type to the current device (which is set using the device command), and gives that device a name and a UUID. This allows us to identify the device for later use, and to know the type of that device.

setup <args...> Types specific device setup commands. For obdfilter, a setup command tells the driver which block device it should use for storage and what type of file system is on that device.

cleanup Cleans up a previously setup device

detach Removes a driver (and its name and UUID) from the current device

lov_setconfig lov-uuid stripe-count default-stripe-size offset pattern UUID1 [UUID2...] Writes LOV configuration to an MDS device

lov_getconfig lov-uuid Reads LOV configuration from an MDS device. Returns default-stripe-count, default-stripe-size, offset, pattern, and a list of OST UUID's.

record cfg-uuid-name Records the commands that follow in the log

endrecord Stops recording

parse config-uuid-name Parses the log of recorded commands for a config

dump_log config-uuid-name Displays the log of recorded commands for a config to kernel debug log

clear_log config-name Deletes the current config log of recorded commands

Device Operations:

probe [timeout] Builds a connection handle to a device. This command is used to suspend configuration until the lctl command ensures the availability of the MDS and OSC services. This avoids mount failures in a rebooting cluster.

close Closes the connection handle

getattr <objid> Gets the attributes for an OST object <objid>

setattr <objid> <mode> Sets the mode attribute for an OST object <objid>

create [num [mode [verbose]]] Creates the specified number <num> of OST objects with the given <mode>

destroy <num> Starting at <objid>, destroys <num> number of objects starting from the object with object id <objid>

test_getattr <num> [verbose [[t]objid]] Does <num> getattrs on an OST object <objid> (objectid+1 on each thread)

test_brw [t]<num> [write [verbose [npages [[t]objid]]]] Does <num> bulk read/ writes on an OST object <objid> (<npages> per I/O)

test_ldlm Performs the lock manager test

ldlm_regress_start %s [numthreads [refheld [numres [numext]]]] Starts the lock manager stress test

ldlm_regress_stop Stops the lock manager stress test

dump_ldlm Dumps all the lock manager states. This is very useful for debugging.

activate Activates an import

deacttivate De-activates an import

recover <connection UUID>

lookup <directory> <file> Displays the information of the given file

notransno Disables the sending of committed transnumber updates

readonly Disables writes to the underlying device

abort_recovery Aborts recovery on the MDS device

mount_option Dumps mount options to a file

get_stripe Shows stripe information for an echo client object

set_stripe <objid>[width!count[@offset] [:id:id....] Sets stripe information for an echo client

unset_stripe <objid> Unsets stripe information for an echo client object

del_mount_option profile Deletes a specified profile

set_timeout <secs> Sets the timeout (obd_timeout) for a server to wait before failing recovery

set_lustre_upcall </full/path/to/upcall> Sets the lustre upcall (obd_lustre_upcall) via the lustre.upcall sysctl.

llog_catlist Lists all the catalog logs on current device

llog_info <$logname|#oid#ogr#ogen> Displays the log header information

llog_print <$logname|#oid#ogr#ogen> [from] [to] Displays the log content information. It displays all the records from index 1 by default.

llog_check <$logname|#oid#ogr#ogen> [from] [to] Checks the log content information. It checks all the records from index 1 by default.

llog_cancel <catalog id|catalog name> <log id> <index> Cancels a record in the log

llog_remove <catalog id|catalog name> <log id> Removes a log from the catalog, erases it from the disk

Debug:

debug_daemon Debugs the daemon control and dumps to a file

debug_kernel [file] [raw] Gets the debug buffer and dumps to a file

debug_file <input> [output] Converts the kernel-dumped debug log from binary to plain text format

clear Clears the kernel debug buffer

mark <text> Inserts marker text in the kernel debug buffer

filter <subsystem id/debug mask> Filters message type from the kernel debug buffer

show <subsystem id/debug mask> Shows the specific type of messages

debug_list <subs/types> Lists all the subsystem and debug types

modules <path> Provides gdb-friendly module information

panic Forces the kernel to panic

lwt start/stop [file] Light-weight tracing

memhog <page count> [<gfp flags>] Memory pressure testing

Control:

help Shows a complete list of commands. help <command name> can be used to get help on a specific command

exit Closes the lctl session

quit Closes the lctl session

Options:
(options that can be used to invoke lctl)

--device The device number to be used for the operation. The value of devno is an integer, normally found by calling lctl name2dev on a device name.

--threads The numthreads variable is a strictly positive integer indicating the number of threads to be started. The devno option is used as above.

--ignore_errors | ignore_errors Ignores errors during the script processing

dump Saves ioctls to a file

4.3.3 Examples

attach

```
# lctl
lctl > newdev
lctl > attach obdfilter OBDDEV OBDUUID
```

connect

```
lctl > name2dev OSCDEV
2
lctl > device 2
lctl > connect
```

getattr

```
lctl > getattr 12
id: 12
grp: 0
atime: 1002663714
mtime: 1002663535
ctime: 1002663535
```

size: 10

blocks: 8

blksize: 4096

mode: 100644

uid: 0

gid: 0

flags: 0

obdflags: 0

nlink: 1

valid: ffffffff

inline:

obdmd:

lctl > disconnect

Finished (success)

setup

lctl > setup /dev/loop0 extN

lctl > quit

The example below shows how to use lctl for viewing the peers that are up:

```
$ lctl > network tcp up

$ lctl > peer_list

12345-ost.cfs.com@tcp [1]client.cfs.com->ost.cfs.com:988 #3

12345-ost2.cfs.com@tcp [1]client.cfs.com->ost2.cfs.com:988 #3

12345-mds.cfs.com@tcp [1]client.cfs.com->mds.cfs.com:988 #3
```

CHAPTER V – 5. LUSTRE PROC ELEMENTS

5.1 Elements in the namespace

The elements in the namespace are struct lprocfs_dentry's. Like normal dentry's, these nodes in the lustre /proc namespace have children and a parent. The children are of 3 types:

- **Regular files** - To indicate statistics associated with the node
- **Subdirectories** - Associated with a major type of instantiation under the node. This will be detailed below.
- **Symbolic Links** - To express well established relations between subsystems in the namespace

For example, an OBD driver type will have a subdirectory node for each device of that type that is instantiated. Such a device subdirectory node may have a symlink to an ldlm namespace directory node, if the device uses one or more namespaces.

When devices connect to other obd devices, an import object is instantiated. The import node can be implemented either as a symbolic link or can be a directory with detailed information underneath. In either case, an import node can point to the /proc node associated with a target device, or can point to a ptlrpc connection that is used to reach a remote target. In the case of exports on servers (OST and MDS server systems) it is not desirable to register all imports or exports in a subdirectory as there are too many.

5.1.1 Top Level Nodes

It is beneficial to have information visible about several Lustre subsystems:

- OBD types
- Meta-data types
- Lustre super blocks
- ldlm namespaces
- ptlrpc connections

The top level /proc/fs/lustre directory will have a subdirectory for each of these subsystems.

5.1.2 Children

For each of the top level types, there are children:

- Meta-data and object driver types
  - Directory node for each MD & OBD device created
    - A subdirectory called namespace on which the file system

can be mounted which holds the objects

- Statistics for Meta-data devices
- A subdirectory with symbolic links to ldlm namespaces
- A subdirectory named imports which contains
- Subdirectories for each export to which a connection is made
    - – A link which is a target to other local devices or ptlrpc connections used
    - – Per export connection data
- A subdirectory named exports which contains
  - – Subdirectories for each import received from a peer
    - – A link which points to a source device of local devices or ptlrpc connections used
    - – Per import connection data

◆ lustre_superblocks

- Symbolic link to the local object device import for file I/O API. The name of the link could be the type of the object device used, like OSC, LOV, COBD, and so on.
- Symbolic link called object_device pointing to the above link
- Symbolic link to the local Meta-data device import used by the file system. The name of the link could represent the type MDC, WBC, CMC.
- Symbolic link called Meta-data device pointing to the above link
- File system statistics

◆ namespaces

- Lock and resource counters
- A directory of imports/exports as described above with statistics as for the parent

5.2 Statistics

5.2.1 Statistics Data Types

Each lprocfs_dentry of directory type can contain statistics entries of a particular type. The type of the statistics data is related to the kind of device described. The statistics types are as follows:

- Call Statistics (These are of following 5 types):
  - OBD call counter
  - MD call counter
  - File system call counter
  - ldlm call counter
  - ptlrpc call counter
- Resource Counters:
  - Meta-data space counter
  - Meta-data space counter
  - Export/Import counter

5.2.2 Statistics Provisioning Methods

The statistics data are filled with a few fundamental methods:

- Call trackers
  - Implemented at the level of method invocations
- Child summaries
  - Implemented exploiting the file system structure in /proc/fs
- Statistics gathering
  - Using cached, reasonably up to date information through statfs style calls

5.3 Implementation

The implementation can now be done with a few simple hooks called at generic points in the code. There should only be code associated with type and device instantiation, and no code specific to any particular device.

Chapter V – 6. System Limits

6.1 Introduction

This section describes various limits on the size of files and file systems. These limits are imposed either by the Lustre architecture or by the Linux VFS and VM subsystems. In a few cases, the limit is defined within the code and could be changed by re-compiling Lustre. In those cases, the limit chosen is supported by CFS testing and may change in future releases.

6.1.1 Maximum Stripe Count

The maximum number of stripe count is 160. This limit is a hard coded option and reflects current tested performance limits. It may be increased in future releases. Under normal circumstances, the stripe count is not affected by ACLs.

6.1.2 Maximum Stripe Size

For a 32-bit machine, the product of stripe size and stripe count (stripe_size * stripe_count) must be less than $2^{\wedge 32}$. The ext3 limit of 2TB for a single file applies for a 64-bit machine. (Lustre can support 160 stripes of 2TB each on a 64-bit system.)

6.1.3 Minimum Stripe Size

Due to the 64KB PAGE_SIZE on some 64-bit machines, the minimum stripe size is set to 64 KB.

6.1.4 Maximum Number of OSTs and MDSs

You can set the maximum number of OSTs by a compile option. The limit of 512 OSTs in Lustre 1.4.6 is raised to 1020 OSTs in Lustre 1.4.7. Rigorous testing is in progress to move the limit to 4000 OSTs.

The maximum number of MDSs will be determined after accomplishing MDS clustering.

6.1.5 Maximum Number of Clients

The number of clients is currently limited to 65536 as defined in the code.

6.1.6 Maximum Size of a File System

In 2.4 kernels, the Linux block layer limits the block devices like hard disks or RAID arrays to 2TB. For i386 systems in 2.6 kernels, the block devices are limited to 16TB. Each OST or MDS can have a file system up to 2TB (The 2TB limit is imposed by ext3 for 2.6 kernels). You can have multiple OST file systems on a single node. The largest Lustre file system currently has 448 OSTs in a single file system (running the 1.4.3 Lustre version). There is a compile-time limit of 512 OSTs in a single file system, giving a single file system limit of 1PB.

Several production Lustre file systems have around 100 object storage servers in a single file system. One production file system is in excess of 900TB (448 OSTs). All these facts indicate that Lustre would scale just fine if more hardware were made available. The 2TB limit on a file system will be soon removed to allow larger file systems with fewer OST devices.

6.1.7 Maximum File Size

Individual files have a hard limit of nearly 16TB on 32-bit systems imposed by the kernel memory subsystem. On 64-bit systems this limit does not exist. Hence, files can be 64-bits in size. Lustre imposes an additional size limit of up to the number of stripes, where each stripe is of 2TB. A single file can have a maximum of 160 stripes, which gives an upper single file limit of 320TB for 64-bit systems. The actual amount of data that can be stored in a file depends upon the amount of free space in each OST on which the file is striped.

6.1.8 Maximum Number of Files or Subdirectories in a Single Directory

Lustre uses the ext3 hashed directory code, which has a limit of about 25 million files. On reaching this limit, the directory grows to more than 2GB depending on the length of the filenames. The maximum number of subdirectories in the versions before Lustre 1.2.6 is 32,000. You can have unlimited subdirectories in all the later versions of Lustre due to a small ext3 format change.

In fact, Lustre is tested with ten million files in a single directory. On a properly-configured dual-CPU MDS with 4 GB RAM, random lookups in such a directory are possible at a rate of 5,000 files /second.

6.1.9 MDS Space Consumption

A single MDS imposes an upper limit of 4 billion inodes. The default limit is slightly less than the device size of 4KB. That means about

512MB inodes for a file system with MDS of 2TB. This can be increased initially at the time of MDS file system creation by specifying the "--mkfsoptions='-i 2048'" option on the "--add mds" config line for the MDS.

For newer releases of e2fsprogs, you can specify '-i 1024' to create 1 inode for every 1KB disk space. You can also specify '-N {num inodes}' to set a specific number of inodes. Note that the inode size (-I) should not be larger than half the inode ratio (-i). Otherwise mke2fs will spin trying to write more number of inodes than the inodes that can fit into the device.

6.1.10 Maximum Length of a Filename and Pathname

This limit is 255 bytes for a single filename, the same as in an ext3 file system. The Linux VFS imposes a full pathname length of 4096 bytes.

Feature List

Task List

Architect tasks

Glossary

A

ACL – Access Control List. An extended attribute associated with a file which contains authorization directives.

Administrative OST failure – A configuration directive given to a cluster to declare that an OST has failed, so that errors can be returned immediately.

C

CFS – Cluster File Systems, Inc., a US corporation founded in 2001 by Peter J. Braam to develop, maintain and support Lustre.

CMD – Clustered meta-data, a collection of meta-data targets implementing a single file system namespace.

CMOBD – Cache Management OBD. A special device which will implement remote cache flushed and migration among devices.

COBD – Caching OBD. A driver which makes decisions when to use a proxy or locally running cache and when to go to a master server. Formerly this abbreviation was used for the word collaborative cache.

Collaborative Cache – A read cache instantiated on nodes that can be clients or dedicated systems, to enable client to client data transfer, enabling enormous scalability benefits for mostly read-only situations. A COBD cache is not currently implemented in Lustre.

Completion Callback – An RPC made by an OST or MDT to another system, usually a client, to indicate to that system that a lock it had requested is now granted.

Configlog – An llog file used in a node or retrieved from a management server over the network with configuration instructions for Lustre systems at startup time.

Configuration lock – A lock held by every node in the cluster to control configuration changes. When callbacks are received the nodes quiesce their traffic, cancel the lock and await configuration changes after which they reacquire the lock before resuming normal operation.

D

Default stripe pattern – Information in the LOV descriptor describing the default stripe count used for new files in a file system. This can be amended by using a directory stripe descriptor or a per file stripe descriptor.

Direct I/O – A mechanism which can be used during read and write system calls. It bypasses the kernel I/O cache to memory copy of data between kernel and application memory address spaces.

Directory stripe descriptor – An extended attribute describing the default stripe pattern for file underneath that directory.

E

EA – See Extended Attribute.

Eviction – The process of eliminating server state for a client that is not returning to the cluster after a timeout or server failures has occurred.

Export – The state held by a server for a client sufficient to recover all in flight operations transparently when a single failure occurs.

Extended attribute – A small amount of data which can be retrieved through a name associated with a particular inode. Examples of Extended Attributes are access control lists, striping information and crypto keys.

Extent Lock – A lock used by the OSC to protect an extent in a storage object for concurrency control of read, write, file size acquisition and truncation operations.

F

Failback – The failover process whereby the default active server regains control over the service.

Failout OST – An OST which when fails to answer client requests is not expected to recover. A failout OST which has failed can be administratively failed, enabling clients to return errors when accessing data on the failed OST without making network requests.

Failover – The process whereby a standby computer server system takes over for an active computers server after a failure of the active node, typically gaining exclusive access to a shared storage device between the two servers.

FID – A Lustre file identifier. A collection of integers which uniquely identify a file or object. The structure contains a sequence, identity and version number.

Fileset –

FLDB – FID Location Database. This database maps a sequence of FID's to a server which is managing the objects in the sequence.

Flight Group – A group or I/O transfer operations initiated in the OSC which is simultaneously going between two endpoints. Tuning the flight group size correctly leads to a full pipe.

G

Glimpse callback – An RPC made by an OST or MDT to another system, usually a client, to indicate to that system that an extent lock it is holding should be surrendered if it is not in use. If the lock is in use the system should report the object size in the reply to the glimpse callback. Glimpses are introduced to optimize the acquisition of file sizes.

GNS – Global Namespace

Group Lock –

Group upcall –

GSS API –

H

Htree – An indexing system for large directories used by ext3. Originally implemented by Daniel Phillips and completed by CFS.

I

Import – The state held by a client to recover a transaction sequence fully after a server failure and restart.

Intent Lock – A special locking operation introduced by Lustre into the Linux kernel. An intent lock combines a request for a lock with the full information to perform the operation(s) for which the lock was requested. This offers the server the option of granting the lock or performing the operation and informing the client of the result of the operation without granting a lock. The use of intent locks leads to even complicated meta-data operations implemented with a single RPC from the client to the server.

IOV – IO vector. A buffer destined for transport across the network which contains a collection, aka as a vector, of blocks with data.

J

Join File –

K

Kerberos – An authentication mechanism, optionally available in 1.6 versions of Lustre as a GSS backend.

L

LAID – Lustre RAID. A mechanism whereby the LOV can stripe I/O over a number of OST's with redundancy. Expected in Lustre 2.0.

LBUG – A bug written into a log by Lustre indicating a serious failure of the system.

LDLM – Lustre Distributed Lock Manager

Lfind – A subcommand of lfs to find inodes associated with objects.

Lfs – A Lustre file system utility named after fs (AFS), cfs (Coda), ifs (Intermezzo).

Lfsck – Lustre File System Check - a distributed version of a disk file system checker. Lfsck normally does not need to be run, except when file systems incurred damage through multiple disk failures and other forms of damage that cannot be recovered with file system journal recovery.

liblustre – Lustre library, a user-mode Lustre client linked into a user program for Lustre fs access. liblustre clients cache no data, don't need to give back locks on time, and can recover safely from an eviction. They should not participate in recovery.

Llite – See Lustre Lite. The word is still in use inside the code and module names to indicate that code elements are related to the Lustre file system.

Llog – A log file of entries used internally by Lustre. An llog is suitable for rapid transactional appending of records and very cheap cancellation of records through a bitmap.

Llog Catalog – An llog with records that each point at an llog. Catalogs were introduced to give llogs almost infinite size. Llogs have an originator which writes records and a replicator which cancels records, usually through an RPC, when the records are not needed.

LMV – Logical meta-data volume, a driver to abstract in the Lustre client that it is working with a meta-data cluster instead of a single meta-data server.

LND – Lustre Network Driver, a code module enabling LNET support over a particular transport, such as TCP, various kinds of InfiniBand, Elan or Myrinet.

LNET – Lustre NETworking, a message passing network protocol capable of running and routing through various physical layers. LNET forms the underpinning of LNETrpc.

Lnetrpc – An RPC protocol layered on LNET. This RPC protocol deal with stateful servers and has exactly-once semantics, and built in support for recovery.

Load Balancing MDS – A cluster of MDS's that perform load balancing of the requests among the systems.

Lock Client – A module making lock RPC's to a lock server and handling revocations from the server.

Lock Server – A system managing locks on certain objects. It also issues lock callback requests calls while servicing or completing lock requests for already locked objects.

LOV – Logical object volume. This is the object storage analog of a logical volume in a block device volume management system such as LVM or EVMS. The logical object volume is primarily used to present a collection of OST's as a single object device to the MDT and client file system drivers.

LOV descriptor – A set of configuration directives which describes which nodes are OSS systems in the Lustre cluster, providing names for their OST's.

LOV Logical Object Volume – An OBD providing access to multiple OSC's and presenting the combined result as a single device.

Lustre – The name of the project chosen by Peter Braam in 1999 for an object based storage architecture. Now the name is commonly associated with the Lustre file system.

Lustre Client – An operating instance with a mounted Lustre file system.

Lustre File – A file in the Lustre file system. The implementation of a Lustre file is through an inode on a meta-data server which contains references to storage object on OSS servers.

Lustre Lite – A preliminary version of Lustre developed for LLNL in 2002. With the release of Lustre 1.0 in late 2003, Lustre Lite became obsolete.

Lvfs – A library providing an interface between Lustre OSD and MDD drivers and file systems, to avoid introducing file system specific abstractions into the OSD and MDD drivers.

M

Mballoc – An advanced block allocation protocol introduced by CFS into the ext3 disk file system capable of efficiently managing the allocation of large (typically 1MB) contiguous disk extents.

MDC – The meta-data client code module which interacts with the MDT using LNETrpc. Also an instance of an object device operating on an MDT through the network protocol.

MDD – A meta-data device, currently implemented using the directory structure and extended attributes of disk filesystems.

MDS – Meta-data Server, referring to a computer system or software package running the Lustre meta-data services.

MDS Client – Same as MDC.

MDS Server – Same as MDS.

MDT – A meta-data target, a meta-data device made available through the Lustre meta-data network protocol.

Meta-data Writeback Cache – Many local and network filesystems have a cache of file data which applications have written but which has not yet been flushed to storage devices. A meta-data writeback cache is a cache of meta-data updates (mkdir, create, setattr, other operations) which an application has performed and which have not yet been flushed to a storage device or server. InterMezzo is one of the first network filesystems to have a meta-data write back cache.

MGS – Management service. A software module managing startup configuration information and changes to this information. Also the server node on which this system is running.

Mount object –

Mountconf – The configuration protocol for Lustre introduced in version 1.6 where formatting disk file systems on servers with the mkfs.lustre program prepares them for automatic incorporation into a Lustre cluster.

N

NAL – An older, obsolete term for LND.

NID – A network id, which encodes the type, network number and network address of a network interface on a node for use by Lustre.

NIO API – A subset of the LNET RPC module implementing a library for sending large network requests, moving buffers with RDMA.

O

OBD – Object device, the base class of layering software constructs that provides the Lustre functionality.

OBD API – See storage object API.

OBD type– Many modules can implement the Lustre object or meta-data API's. Examples of OBD types are the LOV, the OSC and the OSD.

Obdfilter – An older name for the OSD device driver.

OBDFS Object Based File System – A now obsolete single node object filesystem storing data and meta-data on object devices.

Object device – An instance of a object that exports the OBD API.

Object storage – A term referring to a storage device API or protocol involving storage objects. The two most well known instances of object storage are the T10 iSCSI storage object protocol (XXX supply numbers of standards here) and the Lustre object storage protocol. The Lustre protocol is a network implementation of the Lustre object API. The principal difference between the Lustre and T10 protocols is that Lustre includes locking and recovery control in the protocol and is not tied to a SCSI transport layer.

opencache – cache of open file handles. Performance enhancement for NFS

Orphan objects – Storage objects for which there is no Lustre file pointing anymore at these objects. Orphan objects can arise from crashes and are automatically removed by an llog recovery. When a client deletes a file, the MDT gives back a cookie for each stripe. The client then sends the cookie and tells the OST to delete the stripe. The OST finally sends the cookie back to the MDT to cancel it.

Orphan handling – A component of the meta-data service which allows for recovery of open unlinked files after a server crash. The implementation of this features retains open unlinked files as orphan objects until it is clear that no clients are using them.

OSC Object Storage Client – The client unit talking to an OST (via an OSS).

OSD – Object Storage Device. This term is a generic term used in the industry for storage devices with a more extended interface than block oriented devices such as disks. Lustre uses this name for a software module implementing an object storage API in the kernel. Lustre also uses this name for an instance of an object storage device created by that driver. The OSD device is layered on a file system, with methods that mimic the creation, destroy and I/O operations on file inodes.

OSS – Object Storage Server. A system running an object storage service software stack.

OSS Object Storage Server – A server OBD providing access to local OST's.

OST – Object storage target, an object storage device made accessible through a network protocol. An OST is typically tied to a unique OSD which in turn is tied to a formatted disk file system on the server containing the storage objects.

P

Pdirops – A locking protocol introduced in the VFS by CFS to allow for concurrent operations on a single directory inode.

pool – A group of OST's can be combined into a pool with unique access permissions and stripe characteristics. Each OST is a member of only 1 pool, while an MDT can serve files from multiple pools. A client accesses one pool on the the filesystem; the MDT stores files from/for that client only on that pool's OST's

Portal – A concept used by LNET. LNET messages are sent to a portal on a NID. Portals can receive packets when a memory descriptor is attached to the portal. Portals are implemented by as integers.

Examples of portals are the portals on which certain groups of object, meta-data, configuration and locking requests and replies are received.

Ptlrpc – An older term for lnetrpc.

R

Raw operations – VFS operations introduced by Lustre to implement operations such as mkdir, rmdir, link, rename with a single RPC to the server. Other file systems would typically use more operations. The expense of the raw operation is omitting the update of client namespace caches after obtaining a successful result.

Remote user handling –

Replay – The concept of re-executing a request on a server after a server shutdown where the server lost information in its memory caches. The requests to be replayed are retained by clients until the server(s) have confirmed that the data is persistent on disk. Only requests for which a client has received a reply are replayed.

Resent request – Requests that have seen no reply can be re-sent after a server reboot.

Revocation Callback – An RPC made by an OST or MDT to another system, usually a client, to revoke a granted lock.

Rollback – The notion that server state is in a crash lost because it was cached in memory and not yet persistent on disk.

Root squash – A mechanism whereby the identity of a root user on a client system is mapped to a different identity on the server to avoid root users on clients gaining broad permissions on servers. Typically at least one client system should not be subject to root squash for management purposes.

routing – LNET can route between different networks and LNDs

RPC – Remote procedure call, a network encoding of a request**.**

S

Storage Object API – The API manipulating storage objects. This API is richer than that of block devices and includes the creation and deletion of storage objects, reading and writing buffers from/to certain offsets, setting attributes and other storage object meta-data.

Storage objects – A generic notion referring to data containers, similar or identical to file inodes.

Stride – A contiguous logical extent of a Lustre file written to a single object service target.

Stride size – The maximum size of a stride, typically 4MB.

Stripe count – The number of OST's holding objects for a RAID0 striped Lustre file.

Striping meta-data – The extended attribute associated with a file describing how its data is distributed over storage objects. See also default stripe pattern, and directory striping meta-data.

T

T10 object protocol – An object storage protocol tied to the SCSI transport layer.

W

Wide striping – Using many OST's to store stripes of a single file to obtain maximum bandwidth to a single file through parallel utilization of many OST's.

Z

zeroconf – Obsolete from 1.6. A method to start a client without an XML file. The mount command gets a client startup llog from a specified MDS.

Alphabetical Index

L

M

N

O

P

Q

R

S

T

V

W

X

Z

Version Log

Version No.	Details of the changes made	Author	Date
1.1	1) Copyright Page: Changed copyright year from 2004 to 2005, first draft to draft 1.1 2) On page 14: As Nathan requested, heading changed to “upgrading a cluster live” 3) Bookmarks generated by OpenOffice automatically, actually they are generated from table of contents of OpenOffice file.	Eli Li	12/02/05
1.2	1) Cover and Copyright Page: Changed the draft version from 1.1 to 1.2. Changed the date from Dec. 2,2005 to Dec. 9, 2005. 2) Changed the color of all titles from red to black as per the communication between Dr. Braam and Mr. Bojanic. 3) Completed basic editing for Chapter I. 4) Incorporated some of the changes suggested by Dr. Braam and Mr. Bojanic in Chapter I and Chapter II-1.	SPSOFT	12/09/05
1.3	1) Reformatted as per the ORA_Template. 2) Incorporated Dr. Braam and Mr. Bojanic's suggestions wherever possible. 3) Included Eli' ChangeLog and Updated mine. 4) Cover and Copyright page: Updated the date and version. 5) Numbered all the chapters. 6) Generated the TOC.	SPSOFT	12/13/05
1.4	1) Cover Page and Copyright Page: Changed the date and version 2) Drafted and Inserted Part 1 – Chapter 1 A Cluster with Lustre as per ManualProject.mpp 3) ChangeLog: Updated the ChangeLog.	SPSOFT	12/16/05
1.5	Transformed in to the new format	SPSOFT	12/19/05
1.6	1) Changed the version no. and date on Cover Page and Copyright Page 2) Added the Parts 2-5 and their headings making the skeleton complete 3) Added the contents under 1.2 Lustre Server Nodes 4) Enhanced the format further for the complete document and also for the Table of Contents 5) Created 5 diagrams in Visio and pasted them at appropriate places 6) Inserted some of the contents from LustreManual.pdf (developed by Eli) till page 20 7) Converted the ChangeLog page into the VersionLog	SPSOFT	12/23/05

Version No.	Details of the changes made	Author	Date
1.7	1) Inserted a new Cover Page 2) Changed the version no. and date on Cover Page, Copyright Page and VersionLog 3) Deleted 2 previous versions and transformed version 1.4 into the latest format and prepared a Master Document 4) Incorporated majority of the suggestions from Dr. Braam-Mr. Bojanic conversation	SPSOFT	12/28/05
1.8	1) Changed the version no. and date 2) Created a simpler template and format 3) Reformatted all the chapters in the new template and format 4) Incorporated all the review comments received from Cliff 5) Inserted all the new chapters created by SPSOFT at appropriate places 6) Generated a new TOC	SPSOFT	01/31/06
1.9	1) Changed the version no. and date 2) Drafted new Task / Feature Lists 3) Introduced Glossary 4) Introduced various page styles to reflect proper headers and footers 5) Created the master document in a new way as directed in the Writer Manual. 6) Generated an alphabetical Index.	SPSOFT	02/10/06
1.10	1) Changed the version no. and date 2) Updated Task / Feature List 3) Changed the Glossary as given by Peter 4) Changed the entries in TOC and Bookmarks 5) Reformatted the Index	SPSOFT	02/13/06
1.11	1) Changed the version no. and date 2) Updated the glossary as resent by Dr. Braam 3) Changed the attributes of the tables and figures, and their titles for better HTML generation	SPSOFT	02/20/06
1.12	1) Change the version no. and date 2) Changed Part I – Chapter 1 as per Peter's instructions. 3) Replaced POSIX ACLs section in Part IV – Chapter 1. 4) Changed the attributes of figures and tables for better HTML generation 5) Updated the Task and Feature Lists as the references to TOC changed 6) Updated the Index and TOC 7) Enhanced the styles of Headings for better HTML generation	SPSOFT	02/23/06

Version No.	Details of the changes made	Author	Date
1_4_6_manv 1_13	1) Changed the version 2) Mentioned about the registered trademark on the Copyright page 3) Introduced 'About the Manaul' page 4) Moved the Version Log to the end 5) Reshuffled the contents and chapters as instructed by Peter 6) Edited the contents as instructed by Peter 7) Increased the Task List and the Feature List, and introduced Cross references in both of them 8) Hyper-linked the TOC	SPSOFT	03/20/06
Version 1.4.6.1-man-v14	1) Changed the version and date 2) Improved the following chapters - I. Chapter II – 4. Failover II. Chapter III – 1. Lustre I/O Kit III. Chapter IV – 2. Striping and I/O IV. Chapter V – 1. User Utilities (man1) 3) Put the headers and footers in place 4) Incorporated Cameron Harr's comments 5) Inserted the text regarding Trademarks on the Copyright page 6) Changed the cover page	SPSOFT	03/24/06
Version 1.4.6.1-man-v15	1) Changed the version and date 2) Improved following Chapters as per Nathan's suggestions - I. Chapter I-2 – Understanding Lustre Networking II. Chapter II-1 – Configuring Lustre Network III. Chapter II-2 – Configuring Lustre-Examples IV. Chapter II-3 – More Complicated Configurations V. Chapter II-4 – Failover 3) Reformatted the Task/Feature Lists to have auto-generated Page Number References	SPSOFT	03/28/06
Version 1.4.6.1-man-v16	1) Changed the version and date 2) Inserted a new title called Portals LND in the Chapter V – 3. Config Files and Module Parameters 3) Reformatted the Feature List to have the correct reference to Portals LND	SPSOFT	03/31/06

Version No.	Details of the changes made	Author	Date
Version 1.4.6.1-man-v17	1) Changed the version and date 2) Improved following chapters - I. Chapter I-1 – A Cluster With Lustre II. Chapter II-1 – Configuring Lustre Network III. Chapter II-5 – Configuring Quotas IV. Chapter IV-3 – Lustre Security V. Chapter V-1 – User Utilities VI. Chapter V–3 – Config Files and Module Parameters VII. Chapter V-4 – System Configuration Utilities	SPSOFT	04/07/06
Version 1.4.6.1-man-v18	1) Changed the version and date 2) Introduced the chapter - Part2-chapter1-Prerequisites 3) Improved following chapters - I. Chapter II-3 – More Complicated Configurations II. Chapter II-4 – Failover	SPSOFT	04/14/06
Version 1.4.6.1-man-v19	1) Changed the version and date 2) Introduced the chapter - Part3-chapter2-LustreProc 3) Incorporated Tom Fenton's review comments, which included corrections of wrong or misleading content, grammar or formatting mistakes, etc. 4) Updated the LMC commands in Chapter V-4 System Configuration Utilities	SPSOFT	04/21/06
Version 1.4.6.1-man-v20	1) Changed the version and date 2) Changed the contents under Chapter II-6 Configuring Quotas – '6.1.2 Remounting the File Systems' as per the RT # 21218 3) Changed the Title of the Chapter IV-2 – Striping and I/O to Chapter IV-2 – 'Striping and Other I/O Options' as per the RT # 21219 4) Changed the contents under Chapter II-1 – Prerequisites – '1.3.2 Building Lustre' as per the RT # 21229 5) Added the information about how to install Lustre RPMs while upgrading Lustre Chapter II-7 – Upgrading from 1.4.5 in the section '7.1.2 Upgrade a Cluster Using Shutdown' as per the RT # 21229 6) Added Chapter II-8 – RAID as per the RT # 21225	SPSOFT	04/28/06

Version No.	Details of the changes made	Author	Date
Version 1.4.6.1-man-v21	1) Changed the version and date 2) Included information on I/O, Locking and Debug Support in Chapter III-2 – LustreProc 3) Included 'Conventions for Command Syntax' in the front matter	SPSOFT	05/05/06
Version 1.4.6.1-man-v22	1) Changed the version and date 2) Corrected all the links that appear in the manual 3) Converted the utility commands from tabular format to textual format for ease of generating man pages directly 4) Modified the section of fstab from labels to MDS string in Chapter II-6 – Configuring Quotas 5) Added more commands for stopping LNET in Chapter II-2 – Configuring the Lustre Network	SPSOFT	05/12/06
Version 1.4.6.1-man-v23	1) Changed the version and date 2) Added Chapter III-3 – Lustre Tuning	SPSOFT	05/20/06
Version 1.4.6.1-man-v24	1) Changed the version and date 2) Added Chapter IV-4 – Other Lustre Operating Tips 3) Inserted four sample outputs in Chapter V-1 User Utilities	SPSOFT	05/26/06
Version 1.4.6.1-man-v25	1) Changed the version and date 2) Revised the chapters below as per Cliff's instructions - Chapter III – 2 – Lustre Proc Chapter III – 3 – Lustre Tuning Chapter IV – 4 – Other Lustre Operating Tips 3) In Chapter II – 4 – More Complicated Configurations, added notes on LNET lines and –add net option 4) In Chapter V – 4 – System Configuration Utilities, added notes about mountfs option in LMC and LCONF.	SPSOFT	06/02/06
Version 1.4.6.1-man-v26	1) Changed the version and date 2) Edited the section on LCONF utility in Chapter V – 4 – System Configuration Utilities 3) Deleted the material on Proc elements from Chapter III – 2 Lustre Proc and created a new chapter in Part V – Reference as Chapter V – 5 Lustre Proc Elements 4) Added Chapter V – 6 Elevator 5) Updated Chapter II – 6 Configuring Quotas	SPSOFT	06/12/06

Version No.	Details of the changes made	Author	Date
Version 1.4.6.1-man-v27	1) Changed the version and date 2) Deleted Chapter V – 6 Elevator and introduced the information in Chapter II – 1 Prerequisites as 1.4.3 Proper Kernel I/O Elevator 3) Included the 'lfs df' command in the Synopsis section of Chapter V – 1 User Utilities 4) Edited the notes on 'asyncdel' in Chapter V – 4 System Configuration Utilities 5) Introduced information on Liblustre Network parameters in 2.2.2 Module Parameters under Chapter II – 2 Configuring Lustre Network	SPSOFT	06/16/06
Version 1.4.6.1-man-v28	1) Changed the version and date 2) Removed 1.3.2 obd survey from Chapter III – 1 Lustre I/O Kit 3) Added information of LNET_ROUTES setting of Liblustre client in section 2.2.2 Module Parameters of Chapter II – 2 Configuring Lustre Network 4) Inserted the information on order of LNET lines in modprobe.conf in section 4.1.1 Modprobe.conf of Chapter II – 4 More Complicated Configurations	SPSOFT	06/23/06
Version 1.4.6.1-man-v29	1) Changed the version and date 2) Changed the format of examples and the reference of 'RHEL' to 'Red Hat Enterprise Linux v3 Update 3' in Chapter II – 1 Prerequisites – 1.4.3 Proper Kernel I/O Elevator 3) Added a note on loopback interface in Chapter II – 2 Configuring Lustre Network – 2.2.2 Module Parameters 4) Gave the full pathname instead of * in Chapter III – 2 Lustre Proc – 2.2.1 Client I/O RPC Stream Tunables 5) Changed 'portals' to 'lnet' in Chapter III – 2 Lustre Proc – 2.4 Debug Support 6) Corrected the setstripe command in 1.1.1 Synopsis and elaborated information on Stripe Size, Stripe Start and Stripe Count in 1.1.3 Examples in the Chapter V – 1 User Utilities 7) Added a note on loopback interface in Chapter V – 3 Config Files and Module Parameters – 3.2.1.1 Network Topology 8) Added a sample output for invoking lconf on the OST node in Chapter V – 4 System Configuration Utilities – 4.2.3 Examples	SPSOFT	06/30/06

Version No.	Details of the changes made	Author	Date
Version 1.4.6.1-man-v30	1) Changed the version and date 2) Added Chapter II – 9 Bonding 3) Added sample output of peer list for lctl in Chapter V – 4 System Configuration Utilities 4) Added sample output of setstripe and getstripe for lfs in Chapter V – 1 User Utilities 5) Added a note on the pathnames of /proc variables under 2.2.1 Client I/O RPC Stream Tunables in Chapter III – 2 LustreProc. In the same chapter changed 'portals' to 'lnet' all over and '/proc/fs/ldiskfs/xxxx/mb_history' to '/proc/fs/ldiskfs/loop0/mb_history' in 2.2.4 mballoc history 6) Deleted 2 hanging lines and corrected the spelling of passive in 5.1.3 Heartbeat in Chapter II – 5 Failover	SPSOFT	07/11/06
Version 1.4.6.1-man-v31	1) Changed the version and date 2) Added Chapter II - 2. Lustre Installation 3) Edited Chapter II - 1. Prerequisites for proper flow of information considering insertion of Chapter II - 2. Lustre Installation	SPSOFT	07/19/06
Version 1.4.6.1-man-v32	1) Changed the version and date 2) Edited Chapter I – 1. A Cluster With Lustre to improve readability 3) Added information on LNET lnd interface number indexing in 3.2.2 Module Parameters in Chapter II – 3. Configuring Lustre Network 4) Added 9.2 Disk Performance Measurement in Chapter II – 9. RAID 5) Added Chapter V – 6. System Limits	SPSOFT	07/26/06

Version No.	Details of the changes made	Author	Date
Version 1.4.6.1-man-v33	1) Changed the version and date 2) Replaced 'Portals' with LNET in section 2.3 of Chapter I – 2. Understanding Lustre Networking 3) Upgraded the output of llmount.sh in section 2.2.1, and changed the format of the example in section 2.4.2 in Chapter II – 2. Lustre Installation 4) Added sample graphs of Read and Write Performance as section 9.2.1 in Chapter II – 9. RAID Also, edited point no. 10 of section 9.2 for correct description of mdadm in Chapter II – 9. RAID 5) Explained how to set the debug level and added a note in section 2.4 of Chapter III – 2. Lustre Proc 6) Replaced 'Portals' with LNET in section 4.3.2 of Chapter V – 4. System Configuration Utilities	SPSOFT	08/01/06